Universidad Nacional del Sur

Tesis Doctor en Control de Sistemas

Plataformas Electrónicas para la Simulación y el Control de Sistemas Dinámicos no Lineales

Ing. Omar D. Lifschitz

Bahía Blanca Argentina

2016

Prefacio

Esta Tesis se presenta como parte de los requisitos para optar al grado Académico de Doctor en Control de Sistemas de la Universidad Nacional del Sur y no ha sido presentada previamente para la obtención de otro título en esta Universidad u otra. La misma contiene los resultados obtenidos en investigaciones llevadas a cabo en el ámbito del Instituto de Investigaciones en Ingeniería Eléctrica "Alfredo Desages"(UNS-CONICET) durante el período comprendido entre el 1 de Abril del 2008 y 15 de Julio del 2015, bajo la dirección del Dr. Osvaldo Agamenonni y del Dr. Pedro Julián.

Bahía Blanca, 15 de Julio de 2015

Omar David Lifschitz
Departamento de Ingeniería Eléctrica y de Computadoras
UNIVERSIDAD NACIONAL DEL SUR

UNIVERSIDAD NACIONAL DEL SUR
Secretaría General de Posgrado y
Educación Continua

La presente tesis ha sido aprobada el ..../..../...., mereciendo la calificación de ....
(.....)

Resumen

La presente tesis pretende establecer un puente entre el modelado, identificación y control de sistemas y la microelectrónica, aportando una herramienta con la capacidad de afrontar requerimientos de velocidad, precisión y tamaño de implementación. Así mismo pretende, implementar algoritmos de identificación que puedan ser desarrollados en un chip y proveer un análisis de los errores que se producen como consecuencia de la digitalización y operaciones de punto fijo.

Abstract

This thesis aims to establish a bridge from modeling, identification and control systems to microelectronics, providing a tool with the ability to meet requirements of speed, accuracy and size of the implementation. Furthermore, implement an identification algorithm that can be implemented on the chip and provide, also, an analysis of the errors occurring as a result of digitization and fixed point operations.

Agradecimientos

Agradezco...
A mis padres por haberme dado la vida y convertido en la persona que soy. A mis hermanos.
A Paola por su eterno apoyo y paciencia. Y a mis adorados hijos.
A Osvaldo y Pedro por su constante guía a lo largo de todo el trabajo y de escucharme cada vez que los necesité.
A mis compañeros de oficina por haberme acompañado a lo largo de esta travesía.
A todos ellos, les doy las gracias...

Índice general

Capítulo 1

Introducción General

Las estructuras Piecewise Linear (PWL) han sido utilizadas durante las últimas cuatro décadas por su practicidad para representar funciones multivariables con una complejidad reducida. En algunos casos se ha utilizado para reducir la complejidad numérica [1], en otros, ha permitido obtener resultados en tiempo real [2] y en una tercera muestra de casos, para lograr mayor frecuencia de trabajo [3]. Por otra parte, el uso de sistemas no lineales es importante en varios campos de aplicación como la biología [4], modelos de internet para simulación en computadora [5], economía [6] y redes de potencia [7], entre otros.

Además, existen varias áreas donde las funciones PWL han demostrado su eficiencia en la implementación, como es el caso de la resolución de sistemas no lineales en computación, que es el área más tradicional [8, 9, 10]. Es claro que las funciones Look-Up Table (LUT) directas, en donde todos los datos son almacenados en memoria, y las funciones LUT con interpolación lineal, ambas, son consideradas funciones PWL. Se encuentran en la literatura funciones de aproximación que utilizan interpolaciones de orden mayor [11], pero su extensión a dominios multivariables hacen que su implementación sea más compleja desde el punto de vista numérico.

Un área particular donde se han utilizado estas representaciones con éxito es en el diseño de estructuras de cálculo no lineales en paralelo, por ejemplo, para implementar cámaras CMOS con procesamiento de imágenes en el plano focal [12], [13]. Gracias al uso de estas funciones se han logrado implementaciones con desempeño record [14]. e implementaciones 3D [15]. También se han reportado en la literatura procesadores dedicados para la implementación de funciones lineales a tramos con alta resolución [16].

En el área de Electrónica de potencia, estrategias de estimación demandan la evaluación de funciones no lineales con una velocidad de cientos de MHz. Algunos ejemplos son: las aplicaciones de Audio Digital y conversores Analógico Digital que requieren operaciones no lineales para las operaciones de predistorción [17], [18], los convertidores de DC donde la parte no lineal reside en el lazo de control, el convertidor de cc-cc mencionado en [19] que utiliza una LUT de alta velocidad para la implementación de un control Proporcional Integrativo Derivativo (PID) no

lineal, y por último, el control y la estimación de las variables en un motor eléctrico que requiere un volumen considerable de cálculos estáticos (Sistema sin memoria) [20], [21], [22] y justifica el uso de memoria extra en la LUT, para alcanzar el desempeño necesario. Así, la aplicación presentada en [23] usa 64MB Dynamic Random-Access Memory (DRAM).

En el área de Control y Modelado de procesos químicos, el interés de funciones no lineales de algunas variables se ha acrecentado debido a las nuevas tecnologías en los sistemas de control, por ejemplo, el control predictivo basado en modelos [24, 25]. Publicaciones recientes trabajan con frecuencias de muestreo en el orden de los 100KHz, cuyo horizonte temporal está en dos muestras hacia adelante. Si bien las frecuencias de operación del sistema a modelar son relativamente bajas, la predicción tiene que ser rápida de manera de poder realizar el cálculo de la función de costos a minimizar [26, 27] durante el período de muestreo. Existen sistemas donde la función de optimización requiere más de una muestra a futuro [28] para el cálculo de la nueva acción de control. La obtención de estas predicciones a futuro de la plata a controlar resta tiempo de procesamiento a la función de optimización, lo cual implica que el método de predicción tiene que ser lo más rápido posible.

En los sistemas de comunicación, la predistorción digital de banda base [29] permite una buena linealidad y eficiencia de potencia si previamente se compensan las no linealidades del amplificador de potencia. Esto implica el uso de filtros lineales y operaciones no lineales muchas de ellas realizadas con: LUT [30], PWL [17, 31, 32, 33] y redes neuronales de PWL [34].

Por todo lo expuesto hasta aquí, se observa la aplicabilidad actual y futura de las funciones PWL y la clara necesidad de lograr implementar, en un circuito dedicado, este tipo de estructuras PWL, para mejorar el desempeño. La filosofía utilizada se basa en la idea de implementar un Hardware (HW) donde se optimiza el propósito de la aplicación, como es el caso de los Graphic Processor Unit (GPU) y los Digital Signal Processors (DSP).

El desarrollo de este trabajo permitió la interacción de dos grupos de investigación que desarrollan su actividad en la Universidad Nacional del Sur: Grupo de Modelado, Identificación y Control de Sistemas (GMICS) y el Grupo de Investigación en Sistemas Electrónicos y Electromecatrónicos (GISEE). La idea básica era la integración de ambos grupos en un proyecto común para proveer al primer grupo de control, de recursos de HW elaborados en el segundo grupo, de microelectrónica.
El trabajo de esta tesis se basó en el estudio de la implementación de estructuras PWL, remarcando las relaciones de compromiso entre el error de la aproximación y la estructura digital que lo implementa y en analizar los limites realizables de un Application-Specific Integrated Circuit (ASIC) manteniendo la solución óptima entre velocidad, retardo de salida, complejidad de la arquitectura y resultado de la aproximación.

1.1. Organización de la tesis

A continuación se presentan los capítulos que componen este trabajo con una breve descripción de cada uno. El Capítulo 2 presenta cuatro conceptos básicos para la realización de esta tesis: estructuras No Linear Output Error (NOE), estructuras Finite Impulse Response (FIR), las bases existentes para la implementación de las estructuras PWL y un ASIC diseñado en el grupo de microelectrónica.

Una vez presentadas las estructuras NOE, el Capítulo 3, describe la implementación de estas estructuras utilizando el chip PWL, donde se realiza un estudio de los errores de cuantización en la implementación digital. Además, se muestran dos tipos de identificación posibles utilizando las dos bases, destacando las ventajas y desventajas de su aplicación y por último, se presenta un ejemplo utilizando una FPGA para realizar distintas estructuras NOE, y otros, para mostrar los efectos de la digitalización en un conocido algoritmo de identificación.

El Capítulo 4, presenta una estructura PWL para la corrección de un Analog to Digtal Converter (ADC) comercial utilizando datos obtenidos en laboratorio. Se describen los parámetros de un ADC, los inconvenientes de obtener señales con un alto grado de pureza y se hace mención de los problemas de entrenamiento que estos sistemas requieren.
Derivado del estudio de la arquitectura digital, el Capítulo 5, presenta un método para estimar cómo tiene que ser conformada la estructura PWL para acotar los errores de la representación de una función no lineal de R^n. Además, se muestran los lineamientos para definir una arquitectura PWL para una implemetación genérica basada en los errores de cuantización. Se incluyen ejemplos para ilustrar los resultados del método.

A los efectos de mostrar lo tratado anteriormente, se brinda un nuevo diseño en el Capítulo 6, donde se presenta una arquitectura tipo pipeline con el objetivo de mejorar dos aspectos importantes: la tasa de salida de datos y el retardo del sistema o latencia. Este último, de suma influencia en los sistemas realimentados como son las estructuras NOE. Además, se muestra el funcionamiento para las distintas configuraciones posibles: PWL, NOE y FIR.

Por último, en el Capítulo 7, se incluyen las conclusiones generales del trabajo y futuros lineamientos para su continuación.

Capítulo 2

Preliminares

En este capítulo se introducen conceptos básicos necesarios para el desarrollo de la presente tesis. Estos conceptos están directamente relacionados con el proceso de construcción del modelo de un sistema en estudio. Un modelo dinámico de un sistema puede concebirse como una estructura matemática que nos permite describir los vínculos o relaciones de causalidad dinámica existentes entre las variables que afectan al mismo (entradas, perturbaciones, etc.) y las salidas de interés. Contar con un modelo nos permite predecir el comportamiento del sistema original, llevar a cabo acciones de control acordes y analizar al mismo en todo tipo de situaciones previstas.

Existen dos formas de abordar el proceso de obtener el modelo de un sistema: 1. Modelado fenomenológico. Donde se involucra un método analítico, en el que se recurre a leyes básicas de la física para describir el comportamiento dinámico de un fenómeno o proceso. 2. Identificación del modelo. Se trata de un método experimental que permite obtener el modelo a partir de datos reales recogidos del funcionamiento del sistema bajo estudio

Existen muy diversas estructuras de identificación de modelos lineales: Auto-Regressive with eXogenous input (NARX), Autoregressive Moving Average with eXogenous inputs (ARMAX), Output Error (OE), Box-Jenkins (BJ), etc, y sus correspondientes extensiones a no lineales: NARX, NARMAX, NOE y NBJ. En particular en esta tesis se adoptó la estructura NOE, que permite representar un sistema dinámico no lineal de manera acorde para su utilización en muy diversas aplicaciones (predicción, control, simulación, etc). Para implementar la parte estática no lineal de la NOE se utilizaron funciones Lineales a Tramos (LAT o PWL en ingles) las que poseen una serie de ventajas interesantes, además de su fácil realización electrónica, entre las cuales sobresale el hecho de proveer una aproximación uniforme y la facilidad para controlar el sobreajuste [35], [36], [37] y [38]. El factor más importante que motivó el uso de las funciones LAT es las simplicidad de su estructura, que es lineal afín en cada una de las regiones del dominio, lo cual permite reducir sistemas complejos a una colección de problemas lineales.

El inconveniente de las funciones LAT es que se requiere particionar el dominio de las entradas de una manera particular. Existen varias maneras de hacerlo: politópica, simplicial, hiperrect-

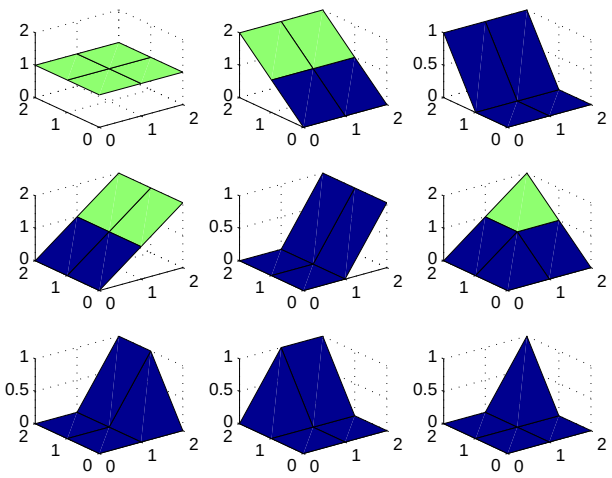

Figura 2.1: Ejemplo de la base estándar para 2 dimensiones con una grilla de 2. Se ve que esta base no es ortonormal y que para un valor cualquiera, todos los parámetros son utilizados para obtener el valor final.

angular y multi-resolución [39]. En este trabajo, se enumeran ventajas y desventajas de cada una de ellas. Desde un punto de vista de implementación digital, se propone utilizar la forma simplicial ya que demanda menos accesos a memoria en el momento de evaluar la función LAT. Esto permite optimizar el diseño para lograr una velocidad de cómputo máxima.

2.1. Representación bases PWL

Entre las maneras posibles de expresar estas funciones PWL, se hizo énfasis en dos posibles bases para su representación [40]: la base estándar y la base general. A modo de ilustración la Fig. 2.1 y Fig. 2.2 muestran cómo se ven ambas bases para un sistema de R2 con una grilla de 2 divisiones.

Como se puede ver en [41], la base estándar no es ortonormal y la evaluación de un punto requiere la participación de todos los parámetros del modelo PWL. Se verá durante el desarrollo del trabajo que esto presenta un inconveniente cuando se quiere implementar digitalmente este tipo de aproximación.
En contrapartida a esta base, existe la base general y la base Gram & Schmidt (Obtenida por el método de G.S.). En dirección a una implementación digital se puede ver que la base general disminuye la cantidad de parámetros en el momento de evaluar la función PWL [40]. Esta particularidad la transforma en la ideal para el desarrollo en un motor PWL. Sin embargo, la base estándar tendrá sus ventajas en el momento de la generalización de los modelos, como se mostrará más adelante.

Las funciones PWL se han usado ampliamente como un núcleo de cálculo para la identificación

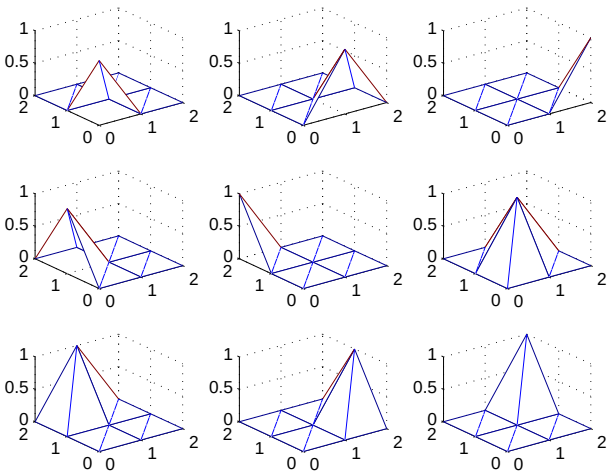

Figura 2.2: Ejemplo de la base general para 2 dimensiones con una grilla de 2. Se ve la ortonormalidad que se verifica observando que vale 1 en el parámetro (o vértice) específico y 0 en el resto de los puntos.

de sistemas no lineales [42, 43, 44], los cuales constituyen una clase interesante de modelos parametrizados Nonlinear Autoregressive Moving Average with eXogenous inputs (NARMAX) [45, 46, 47, 48].

2.2. Estructura NOE

El modelo NOE se basa en el modelo OE aplicado a sistemas no lineales. Este, junto con el NARX y ARMAX, es una de las estructuras más utilizadas. Además, es el modelo más simple de la clase de modelado con error de salida. Su forma general es la siguiente:

$$y(k) = \frac{B(q)}{F(q)}u(k) + v(k) \tag{2.1}$$

Donde $v(k)$ es el ruido aditivo y q indica el operador de desplazamiento hacia delante ($q^{-1}x(k) = x(k-1)$). El predictor óptimo del OE es de hecho un simulador porque utiliza mediciones de la salida del proceso y(k) para obtener la muestra futura. Su forma es la siguiente:

$$\widetilde{y}(k) = \frac{B(q)}{F(q)}u(k) \tag{2.2}$$

Una manera de ver porque no es lineal en sus parámetros es desarrollando la Ec. 2.3.

$$\widetilde{y}(k) = b_1u(k-1) + \ldots + b_mu(k-m) - f_1\widetilde{y}(k-1) - \ldots - f_m\widetilde{y}(k-m) \tag{2.3}$$

La salida del predictor $\widetilde{y}(k-i)$ depende de los parámetros del modelo y, por su parte, el término $f_i\widetilde{y}(k-i)$, también. Esta doble dependencia hace que no sea lineal en obtención de los parámetros. Este efecto se observa, también, en el hecho de la recursividad implícita en la estructura.

Es claro que el predictor es inestable, si el polinomio F(q) lo es, por lo tanto los OE no pueden ser usados para modelado de procesos inestables. Los parámetros del OE pueden ser identificados por una optimización no lineal o por un proceso repetitivo de mínimos cuadros y filtrado correspondiente.

El proceso de identificación en el dominio del tiempo se basa en la evaluación de las relaciones entre los datos de entrada-salida y salidas futuras. Con un número finito de entradas y últimas salidas almacenados en un vector de regresión, el problema principal radica en la evaluación de la función que relaciona el vector de regresión con la salida siguiente.

Existen varias técnicas para el caso en que la función es lineal, pero, es el no lineal, el que presenta el problema más difícil debido al número de funciones diferentes que se pueden utilizar. Los sistemas dinámicos no lineales con un elemento autorregresivo pueden ser implementados de dos maneras diferentes: la primera se conoce como modelo Nonlinear Auto-Regressive with eXogenous input (NARX), mientras que la segunda se conoce como modelo NOE.

El modelo NARX realiza una predicción de un solo paso, es estable y los parámetros son fáciles de evaluar ya que la estructura del modelo no se realimenta. En cambio el modelo NOE, se basa en los datos pasados y presentes, y aunque sufre del consumo de tiempos en identificación y problemas de estabilidad debido a la realimentación inherente en la estructura, tiene el mejor rendimiento en la predicción de un número arbitrario de muestras de salida en el futuro. Esto sería para el caso del control predictivo, detección de fallos, simulación.

2.3. Estructura FIR

La estructura FIR es el modelo lineal más simple debido a que su salida es la suma ponderada de las entradas anteriores al sistema. Como el ruido blanco entra al modelo en forma aditiva, se considera al FIR de la clase de los modelos "Output Error". La motivación de los modelos FIR proviene del hecho de que la salida de cada sistema lineal se puede expresar mediante la suma de convolución:

$$y(k) = \sum_{i=1}^{\infty} g_i u(k-i) = g_1 u(k-1) + g_2 u(k-2) + \dots \tag{2.4}$$

donde cada g_i es la respuesta al impulso. El término $g_0 u(k)$ no se incluye porque se asume que no existe, dando lugar a un camino de alimentación de paso directo desde la entrada a la salida (Sistema estrictamente propio). Es claro que el modelo FIR es una aproximación de esta suma infinita (2.4) ya que los términos en el FIR son finitos. Esto es posible ya que en los sistemas estables los coeficientes g_i tienden a 0 cuando $i \to \infty$. Es por eso, que los modelos

FIR representan sistemas estables. (No obstante, pueden representar a procesos inestables en las primeras muestras para respuestas impulsivas o de tipo escalón). Por el otro lado, la estabilidad inherente de los modelos FIR en comparación a otros es una ventaja cuando el sistema a modelar es realmente estable. El modelo FIR es lineal en los parámetros, por lo tanto, los coeficientes pueden ser estimados utilizando mínimos cuadrados. Una desventaja importante de los modelos FIR es que el orden de los términos de la Ec. 2.4 tiene que ser grande para poder capturar todos los términos g_i que son significativamente diferente de 0, sino el error será grande y la dinámica de la representación será pobre. Análogamente, existe el Nonlinear Finite Impulse Response (NFIR) donde la función que se utiliza es no lineal.

2.4. ASIC PWL

En el grupo de investigación GISEE, se diseñó un PWL ASIC programable para aproximar funciones de hasta 6 dimensiones de entrada y una de salida. Es decir: $R^1 \dots R^6 \rightarrow R^1$ [16].

El chip mencionado fue una prueba de concepto realizado mediante un diseño top-down utilizando el flujo de diseño Electronic Design Automation (EDA). Su núcleo estaba consolidado por una estructura microprogramada. Esta característica permitía las opciones de elegir el número de entradas al chip. El diseño digital presentaba cuantizaciones en las variables de entrada, salida y en los parámetros propios del modelo que eran almacenados en un dispositivo de memoria estándar.

Este ASIC fue la base para el desarrollo de la aplicación de la estructura NOE. Además, se realizó un estudio de errores propios de la digitalización y técnicas de validación en laboratorio. (Apéndice A).

A continuación se presenta una breve reseña de la arquitectura del chip implementado y se muestra la cuantización que se realizó a las variables de entrada y salida del chip. Esta cuantización y operaciones de punto fijo se utilizó para el estudio de los errores como consecuencia de la digitalización.

Arquitectura

La Figura 2.3[1] presenta el esquema de la arquitectura del chip. Una descripción más detallada y todo lo referente a la puesta en marcha y validación del chip se encuentra en el Apéndice A

Cuantización:

La Figura 2.4 presenta la composición en bits de las entradas y salidas del chip. Además, muestra la memoria interna y la memoria externa donde se almacena el código y los parámetros

[1] Cortesía del Dr. Agustín Rodríguez

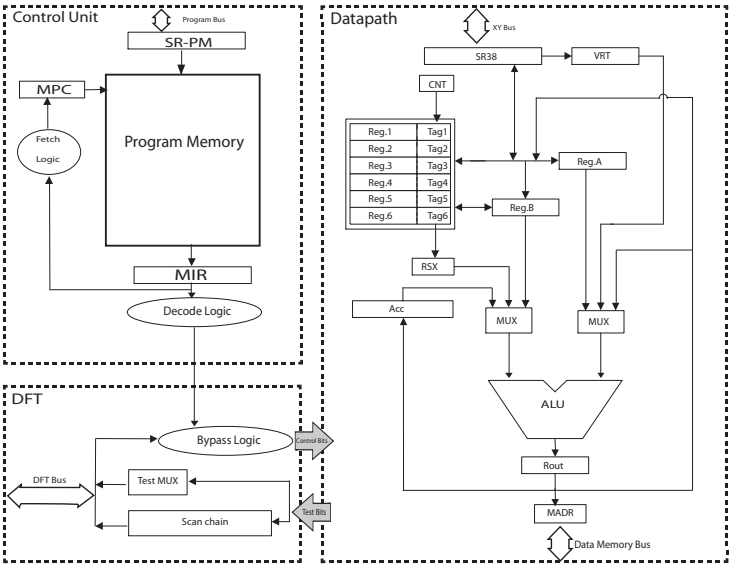

Figura 2.3: Esquemático de la arquitectura del ASIC. Representación de los bloques funcionales del chip.

del modelo respectivamente.

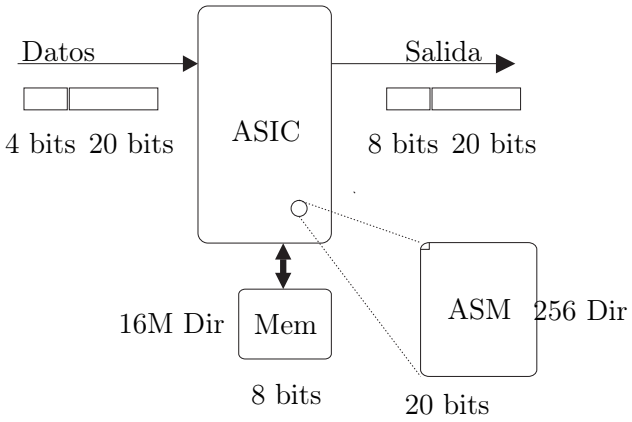

Figura 2.4: Se puede ver la cantidad de bits involucrados en cada entrada y salida del chip. Además, se puede ver la cuantización de la memoria y la cantidad de bits de la memoria interna de código. Las entradas tienen 4 bits de parte entera y 20 bits de parte fraccionaria.

Las entradas tienen 4 bits de parte entera y 20 bits de parte fraccionaria. La salida tiene 8 bits de parte entera y 20 de parte fraccionaria. Los parámetros del modelo PWL están cuantizados en 8 bits.

Capítulo 3

Estructuras PWL-NOE

3.1. Introducción

Durante el desarrollo de este capítulo se presenta la implementación de una estructura NOE utilizando como motor de cálculo una función PWL. La idea principal de utilizar una estructura NOE es la de identificar un sistema conociendo sus entradas y salidas. Es decir, sin tener el total conocimiento del modelo matemático involucrado.

Una manera de obtener los datos es utilizando técnicas de muestreo mediante sensores de magnitudes físicas para su transformación a señales de tensión o corriente. Ya que estos datos se toman en tiempo real, existen dos formas de procesarlos: "Batch" u "On-line", donde la diferencia radica en si los datos son almacenados y luego procesados o el procesamiento se realiza por cada dato que sale de la planta en tiempo real. Es claro que ambas tienen ventajas y desventajas. Su impacto en los algoritmos de identificación se analizará más adelante.

Luego se describe el algoritmo de identificación y sus fundamentos teóricos. Se agregan las dos maneras posibles de aplicar el algoritmo para la actualización de los parámetros del modelo para los datos tipo "Batch" y se detallan las ventajas de cada alternativa.

Debido a que esta estructura NOE es implementada digitalmente, se analizan los errores como consecuencia de la resolución finita. Además, se pone de manifiesto el error de realimentación que existe en la estructura NOE y cómo este afecta la cuantización de los parámetros del modelo.

Más adelante, se muestran tres ejemplos de implementación de diferentes estructuras utilizando el chip donde la identificación se realizó utilizando Matlab y los parámetros del modelo fueron almacenados en la memoria de la estructura. En estos ejemplos se evidencian los errores de cuantización, truncación, aritmética de punto fijo y realimentación.

Hacia el final se presenta una identificación realizada en FPGA donde se marcan los efectos de la digitalización en el resultado del proceso de identificación y se evidencian las velocidades obtenidas para el dispositivo FPGA. Finalmente, se incluyen las conclusiones pertinentes del capítulo.

3.2. Algoritmo de identificación

Definición del Error entre el modelo NOE y la planta a identificar

En esta sección se describe un algoritmo de identificación de los parámetros del modelo NOE. El algoritmo de identificación es conocido en la literatura como "Method of Steepest Descent" o "Algoritmo del gradiente". Este algoritmo es aplicado en las bases: estándar y general, donde se verá las ventajas y desventajas de cada una.

Para poder aplicar el algoritmo de identificación, primero, se tiene que definir el error como la diferencia entre el modelo NOE y el modelo del sistema que se quiere identificar, como muestra la Fig. 3.1.

Las condiciones iniciales del modelo NOE se toman con un valor 0. De hecho, esto no altera los resultados de la identificación.

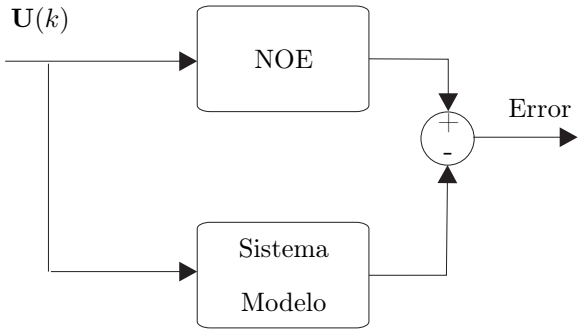

Figura 3.1: Definición del error entre el modelo NOE y el sistema de caja negra que se quiere identificar.

El diagrama en bloques del sistema de identificación se muestra en la Fig. 3.2.

A continuación se detallan las ecuaciones que involucran el proceso de identificación, este proceso aplica el algoritmo del gradiente. Obtenido el valor de salida del modelo NOE y del sistema de caja negra definimos el error instantáneo según la Ec. 3.1

$$e_i = \frac{1}{2}[y_i - \tilde{y}_i]^2 \tag{3.1}$$

Donde $\tilde{y}$ es la salida del modelo NOE e y es la salida del sistema real.

La Ecuación 3.2 describe la salida del bloque PWL

$$\tilde{y} = \sum_{k=1}^{R} c_k \gamma_k \tag{3.2}$$

Donde c son los parámetros que están almacenados en la memoria externa, γ es la diferencia de las partes fraccionarias de las entradas ordenadas de menor a mayor y $R = Dim + 1$. (Se define Dim como la dimensión del espacio de entrada de la función PWL).

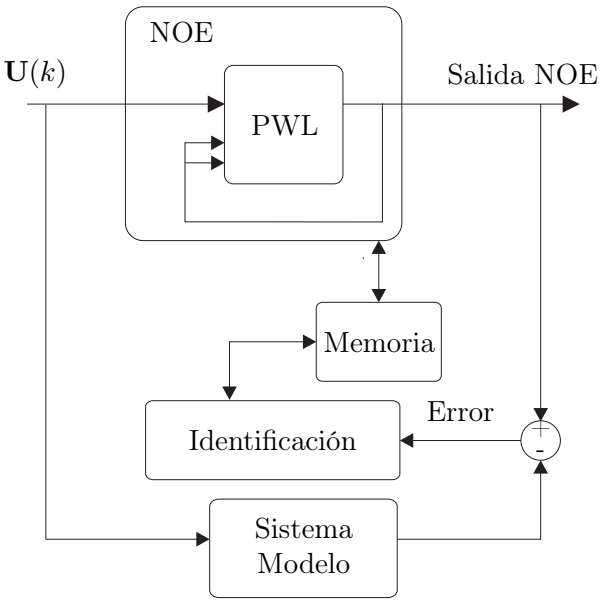

Figura 3.2: Diagrama en bloque del algoritmo de identificación. Estos bloques están implementados en una FPGA.

Usando Ec. 3.2 y Ec. 3.1 el error queda expresado por la Ec. 3.3

$$e_i = \frac{1}{2}[y_i - \sum_{k=1}^{R} c_k \gamma_k]^2 \tag{3.3}$$

Derivando respecto de cada parámetro c en la expresión del error, este queda según Ec. 3.4

$$\begin{aligned} \frac{de_i}{dc_k} &= (y_i - \sum_{k=1}^{R} c_k \gamma_k)\gamma_k \\ &= e_i \gamma_k \end{aligned} \tag{3.4}$$

Tomando el gradiente a la función error:

$$\bigtriangledown e_i = [\frac{de_i}{dc_1}, \frac{de_i}{dc_2}, ..., \frac{de_i}{dc_n}] \tag{3.5}$$

Donde el índice n es el número total de parámetros. El gradiente es ralo, por lo tanto si tenemos un punto x que pertenece a un simplice habrá "Dim + 1" componentes distintos de 0. Esto se expresa en la Ec. 3.6

$$\begin{aligned} \frac{de_i(x)}{dx_j} &= e_i\gamma_j = e_i\mu_j \quad v = \{1, ..., n+1\} \\ \frac{de_i}{dx_j} &= 0 \quad \forall j : j \notin v \end{aligned} \tag{3.6}$$

Donde v es el conjunto de parámetros de un símplice específico que corresponde al punto x.

Finalmente la Ec. 3.7 muestra como se ajustan los parámetros del modelo.

$$C^{k+1} = C^k - \bigtriangledown e_i^k(x)\varphi \tag{3.7}$$

Donde φ es un factor externo que permite elegir el peso que se le atribuye a cada dato de entrada. Este parámetro controla la velocidad de convergencia y como el algoritmo va a ajustar los datos de entrada - salida.

La Fig. 3.3 muestra el diagrama de flujo del algoritmo. El mismo cuenta con dos controles importantes: la cantidad de datos de entrada - salida que se quieren procesar y la cantidad de iteraciones que se quiere reiterar sobre esos mismos datos.

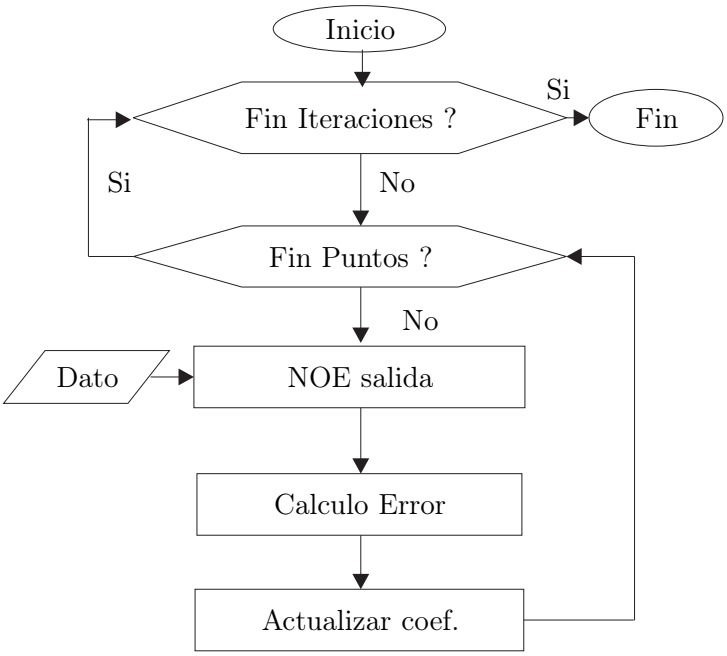

Figura 3.3: Flujo del algoritmo de identificación. Se observan dos controles importantes: Cantidad de datos entrada - salida que se incluyen en el proceso y el número de iteraciones que se quiere reiterar estos datos.

3.3. Implementación de la identificación

Descripción del Bloque de Identificación

A continuación se describe el bloque de identificación realizado sobre una plataforma de FPGA, que se implementó usando una máquina de estados con dos estados generales y cinco estados por cada parámetro de memoria que se tiene que actualizar, como muestra la Fig. 3.4.

Para el caso de tres dimensiones, la cantidad de estados de la maquina fueron veintidós ya que hay cinco estados que se repiten cuatro veces. (Se planteó una arquitectura para la identificación de forma paralela, que se muestra en el Apéndice B, que reduce la cantidad de estados, pero esta no se implementó en la FPGA). El procesamiento de este bloque se realiza una vez finalizado el calculo PWL.

El error, entre el modelo y la aproximación PWL, se calcula en uno de los estados generales y luego se usa en cada uno de los parámetros. El valor μ y las direcciones de memoria que intervinieron en el cálculo PWL se obtienen durante la ejecución del mismo. Estos valores almacenados son parte de las entradas para el bloque de identificación.

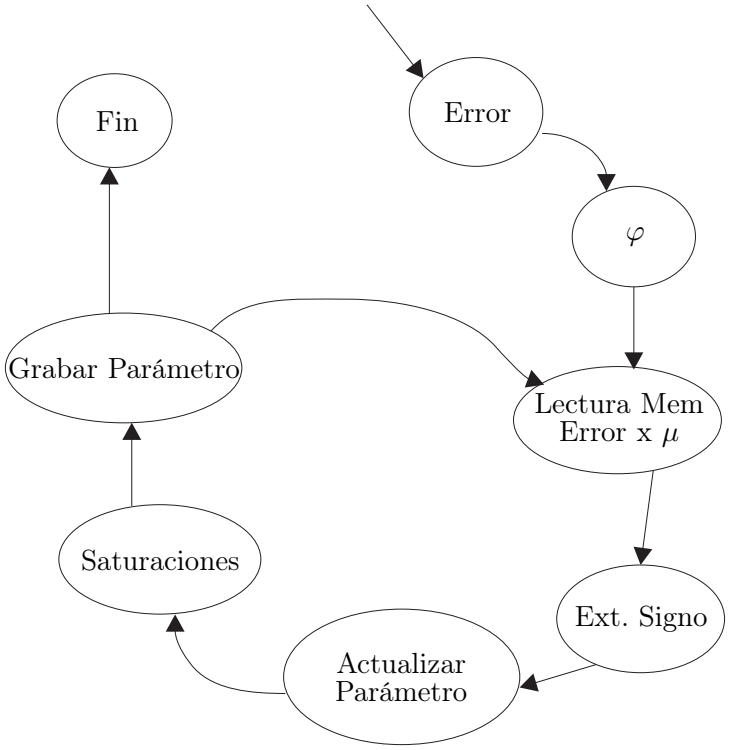

Figura 3.4: Diagrama de los estados para el algoritmo de actualización de parámetros de memoria. El ciclo interno se repite la cantidad de direcciones que hay que actualizar, en este caso se repite 4 veces.

La saturación ocurre cuando el parámetro resultante está fuera del rango permitido. Si esto ocurre, el valor por defecto del parámetro será cualquiera de los extremos del rango según esté por arriba o por debajo del limite. Para el caso concreto los limites de saturación fueron [xEF 0].

Este estado de saturación tiene su importancia ya que los parámetros del modelos no están inicializados en un valor en particular y al ser un sistema realimentado, el error se puede propagar dando valores fuera del rango. Por otro lado, la funcionalidad de la saturación esta estrechamente ligado al factor φ de la Ec. 3.7 ya que cuanto más pequeño sea este factor menos actuación tendrán las funciones de saturación.

Tipos de datos entrada-salida

Dentro del análisis de implementación de una estructura NOE se plantearon dos posibles formas de obtener los datos de entrada-salida de la planta o sistema para la cual se quiere realizar la identificación. Las formas son las siguientes:

- Batch: Este caso se presenta cuando los datos de entrada y salida del sistema a identificar fueron ya adquiridos y se encuentran almacenados en un vector de datos. Por lo tanto la identificación del sistema se hace utilizando los datos previamente almacenados. En esta situación es muy común iterar los datos mientras se aplica un método para el cálculo de los parámetros del modelo. Se verá más adelante como es la condición de salida de esta íteracion. Una ventaja implícita de este forma es que si un dato fue alterado por algún tipo de error, este dato pude ser eliminado fácilmente del vector.

- On-line: Este método utiliza datos provenientes del sistema que son adquiridos con éste en funcionamiento. Es decir, la identificación se produce en paralelo al funcionamiento del sistema. La ventaja innata de esta forma es que no requiere almacenar los datos ya que una vez que el dato actual aportó al modelo, es descartado. En contra parte a la forma Batch, si un dato es erróneo, igualmente, es utilizado para la identificación ya que no se puede determinar la validez de los datos de entrada por ocurrir estos en tiempo real. Se sabe que de la forma "On-line" se puede obtener la forma "Batch". Una de las ideas que sustenta la forma On-line es cuando la identificación se tiene que hacer con el sistema en funcionamiento ya que éste presenta una dinámica propia o forzada por una acción de control.

Formas de actualización de los parámetros

Una vez determinado cómo se pueden obtener los datos de entada-salida, se desarrollan dos posible formas de actualizar los parámetros del modelo. Estas formas se denominaron:

- Identificación Instantánea:

- Identificación Acumulativa:

Para facilitar la definición de estas formas de identificación, se utiliza la Fig. 3.5 donde se muestra un ejemplo de un PWL de R2. La Figura muestra los parámetros y los simplices.

Suponemos un recorrido cualquiera de la planta y con una cierta cantidad de datos, marcadas con un cruz, que caen en forma aleatoria sobre los simplices. De esta manera, cada simplice tiene una cierta cantidad de datos. Además, cada parámetro contiene cuántos puntos van a influir en la identificación propia de éste. La cantidad de puntos está marcada entre paréntesis.

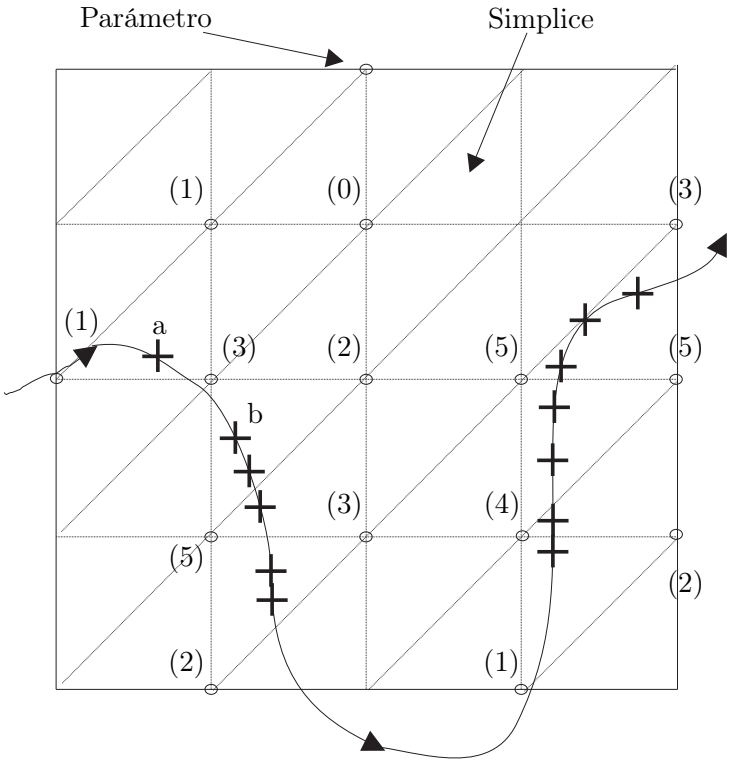

Figura 3.5: Actualización de los parámetros de memoria. Entre paréntesis se indica la cantidad de veces que el vértice particular es actualizado según el recorrido de los datos de entrada. Las cruces indican los datos que contiene cada simplice. Este caso es para una sistema de dimensión 2

A continuación se definen las dos formas de actualización:

Identificación Instantánea: Para cada par de datos entrada-salida se hace el ajuste del parámetro. Es decir, para el punto "a" de la Fig. 3.5 se modifican los 3 parámetros del simplice. (Se recuerda que para un R2 corresponden tres parámetros) Cuando se ingresa el punto "b", este provoca el ajuste de dos nuevos parámetros del simplice vecino y un parámetro en común al punto "a". Es interesante mencionar que el resultado del cálculo PWL para el punto "b" se realiza con dos parámetros con el valor de defecto y un parámetro modificado por el punto "a" anterior. Si la identificación es "On-line", ésta es la única manera posible de identificación.

Identificación Acumulativa: El ajuste de parámetros de la estructura PWL se realiza una vez que se recorrió todo el vector de datos de entrada-salida. Es decir, durante el recorrido se calculan los gradientes que se obtienen por cada dato y esto se almacena. Luego, el ajuste de los parámetros se hace con el promedio de los gradientes que se obtuvieron. En referencia a la Fig.

3.5 ocurre que el valor de la función en los puntos “a” y “b” es calculada por el PWL utilizando los parámetros por defecto cuando se inicializó el sistema. Luego de cada iteración del vector de datos, los parámetros del modelo se ajustan.

Ventajas y desventajas de las formas de identificación

Existiendo la posibilidad de utilizar ambas formas de identificación se tiene que evaluar cuándo conviene utilizar la identificación instantánea y en que situación, la acumulativa.

La implementación digital acumulativa requiere del doble de memoria a los efectos de mantener el promedio de los gradientes y la cantidad de parámetros. Una vez finalizado el vector de datos de entrada-salida se procede a la actualización de los parámetros, para esto se requiere hacer por cada parámetro: una suma, 2 lecturas y 1 escritura a memoria. Ya que, de esta manera se lee el parámetro y el vector de gradiente correspondiente y luego se hace la actualización realizando la suma para, finalmente, realizar la escritura del parámetro modificado a memoria. Hay que remarcar que esto es por cada ciclo de iteración en que se quiere correr el algoritmo de ajuste.

Por otro lado, la identificación acumulativa presenta inmunidad al ruido ya que utiliza el promedio de los gradientes de todos los datos que aportan a un parámetro antes de hacer el ajuste final del parámetro, por lo tanto, el ruido aditivo se cancela.

Para poder mostrar este efecto del ruido aditivo y la bondad de la forma acumulativa, se realizó en Matlab una simulación donde se agregó ruido gaussiano de diferentes σ (σ^2) en la salida del sistema y luego se utilizaron estos datos para efectuar el proceso de identificación con ambas formas.

El criterio utilizado para evaluar los resultados fue calcular el valor Error Cuadrático Medio (ECM) de lo obtenido en validación para ambos métodos. Los resultados se presentan en el Cuadro 3.1. Se utilizaron 3000 datos de entrada - salida para identificación y se realizaron 180 iteraciones. Para la parte de validación se tomaron 100 datos y se utilizó la salida del sistema real sin ruido aditivo.

Se puede decir que el desempeño de la forma acumulativa es mejor ante la presencia de ruido en el sistema pero su implementación digital requiere más memoria y capacidad de cálculo.

Existe otro aspecto que apunta a la identificación Acumulativa como favorita cuando se requiere variar en forma dinámica el factor de ajuste del parámetro (Ec. 3.7). Se puede variar de dos maneras: una, modificando el coeficiente φ y la otra, agregando una variable conocida como “Momento” [49].

El factor multiplicativo φ se relaciona con la velocidad de convergencia del proceso de identificación y, el Momento, asegura la búsqueda de un mínimo global. Estos dos elementos no se pueden considerar en la identificación instantánea ya que la decisión de cambio no se puede hacer utilizando solamente muestras puntuales. En cambio, la identificación acumulativa utilizaría todos los datos de entrada - salida para tomar una decisión más conservativa.

Cuadro 3.1:
Valores del ECM obtenidos en las etapas de validación para entrenamientos con diferentes σ de ruido gaussiano utilizando los dos métodos de identificación. Se observa como el acumulativo es inmune al ruido aditivo.

σ	Instantáneo	Acumulativo
0.0	0.011	0.015
0.1	0.109	0.066
0.2	0.175	0.092
0.3	0.269	0.1161

Consideraciones de la implementación digital

En este apartado se enuncian primeramente algunas de las consideraciones más importantes de la implementación digital para luego avanzar sobre los ejemplos de dicha implementación.

Aritmética de Punto Fijo

En el chip desarrollado, por el grupo GISEE, todos los parámetros del modelo, incluyendo las entradas y salidas de la estructura NOE son positivos pero cuando se implementa la identificación, ésta requiere el cálculo de un error y éste puede ser negativo o positivo. Debido a esto, se trabaja con una librería de signados en la descripción de HW de alto nivel y entonces todos los números son considerados con signo. A continuación se explica la transformación que se realizó para adaptar los nuevos formatos numéricos.

A todos los números se les dio un bit de signo de manera de respetar su cuantización original. Por otro lado, se agregaron los bits necesarios para poder multiplicar y sumar números mayores y menores que 1 respetando el punto decimal. Esto se debe a que los μ son menores que 1 y todo los números son mayores que uno. Estos cambios se muestran en la siguiente ecuación:

$$\begin{aligned} e &= S[7-0], 00000000 \\ Parametro &= Sxxxxxxxx \\ \mu &= S00000000, [20-0] \end{aligned} \tag{3.8}$$

Donde al error (e) se le agregó un bit de signo y 8 bits de parte fraccionaria. A este error se le aplicó el factor φ haciendo un corrimiento de los bits hacia la derecha y completando los bits de la izquierda con la extensión del signo. El corrimiento hacia la izquierda se hizo para no utilizar un divisor en HW. Los coeficientes μ tienen sus 20 bits de parte fraccionaria y se les agrego 7 bits de parte entera en valor 0 y el bit de signo en 0 ya que los μ son positivos. El parámetro es el valor que se obtuvo de la memoria y es positivo, por lo tanto se le agrego un bit en 0, a la izquierda.

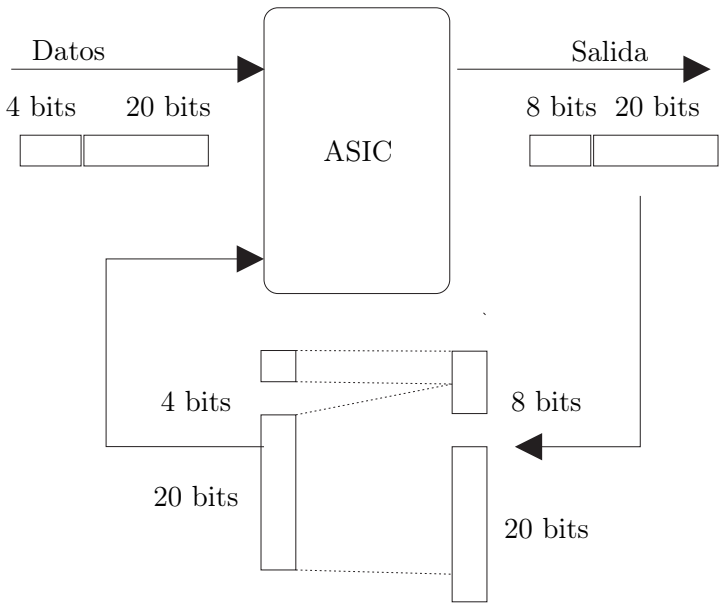

Figura 3.6: Se muestra como se tiene que truncar la salida del chip de manera de poder ajustar la señal al rango de entrada.

Vale la pena mencionar que cada resultado de las operaciones se fue truncando de manera tal que minimizara la complejidad de la arquitectura. Esto es debido a que, en esta implementación en particular, el parámetro es de 8 bits, por lo tanto, no tiene sentido mantener y realizar las operaciones con 8 bits o más para luego realizar el truncamiento final a 8 bits.

Error de la Realimentación

A continuación se presenta el error que se introduce por la realimentación natural de las estructuras NOE. Según este modelo se necesita realimentar las salidas del PWL a la entrada. En este caso tenemos que ajustar el rango de salida al rango de entrada del PWL. Es aquí donde se produce un truncamiento de la salida cortando los últimos 4 bits menos significativos quedando como parte entera los 4 más significativos de la salida. Este truncamiento (Fig. 3.6) tiene un efecto importante en el rango de valores de los parámetros de memoria. El rango de entrada corresponde al intervalo [0 15). Esto significa que el mayor valor posible es el: 0xE,FFFF donde tenemos 4 bits de parte entera y 20 bits de parte fraccionaria. Por otro lado, el mayor valor de la salida es 0xFF,FFFF, ya que son 28 bits. Como se dijo anteriormente, el valor de la salida realimentado será 0xF,FFFF pero este valor esta fuera del rango. Una manera de manejar este error es limitando el rango de los parámetros de memoria a un valor máximo de 0xEF. De esta manera, el mayor valor de salida será 0xEF,FFFF y el truncamiento dará como resultado 0xE,FFFF que está dentro del rango permitido para la entrada. En resumen, la realimentación natural del modelo NOE hace que el rango de los parámetros de memoria esté limitado a [0 0xEF].

3.4. Ejemplos prácticos de la implemetación en un ASIC

Ejemplos implementados en el chip PWL

Esta parte presenta una serie de tres ejemplos que involucran distintos sistemas, cuya identificación requirió de modelos NOE que aumentaban en complejidad. Las estructuras PWL son implementadas en el chip y se utiliza una FPGA para su control. La FPGA se encarga de: entregar los datos de entrada al chip PWL para que este realice el cálculo, luego se encarga de recibir el resultado del chip para pasar los datos hacia la Personal Computer (PC) para la interfase con el usuario. (Más información del protocolo maestro-esclavo se encuentra en el apéndice A).

Los parámetros del modelo se obtienen utilizando un algoritmo desarrollado en Matlab [49]. Este algoritmo implementa el "Method of Steepest Descent" partiendo de una aproximación lineal y luego se incrementa el número de divisiones del modelo PWL usando el anterior como punto de partida. Este algoritmo devuelve los parámetros para la base estándar y entonces queda obtener los parámetros del modelo para la base general. Como las dimensiones y la grilla no se alteran entre una base y la otra, esto da una relación unívoca entre los parámetros. Por lo tanto, se obtienen los parámetros en la base general usando la estimación de la base estándar.

Para facilitar la inicialización de la memoria del sistema con los parámetros del modelo, se ordenaron los parámetros de manera de recorrer en orden creciente las direcciones de la misma.

La Figura 3.7 muestra la configuración práctica del ASIC y de la FPGA. La PC se usa para activar la FPGA, inicializar la memoria y obtener los datos de salida.

Figura 3.7: Foto del ensayo práctico del ASIC y de la FPGA. Se usa la computadora para activar la FPGA y recibir los datos finales.

La Figura 3.8 presenta el esquema del ensayo práctico.

Con el fin de mostrar el comportamiento numérico del ASIC y comparar los resultados vamos

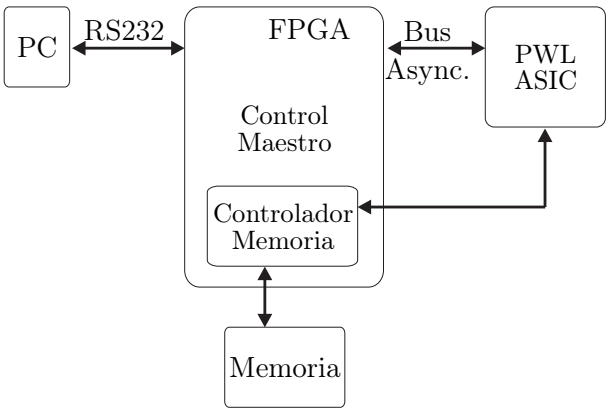

Figura 3.8: Esquema de configuración del ASIC y de la FPGA.

a considerar otros tres modelos, además de la estructura representada por el ASIC. Por lo tanto los cuatro modelos serán los siguientes:

- **Sistema Real:** es el sistema original no lineal implementado en Matlab con la precisión de punto flotante de 64-bit. Este modelo está representado por una ecuación analítica y representa el modelo físico.
- **PWL Ideal:** se basa en una aproximación PWL de la ecuación no lineal original implementada en Matlab. La precisión utilizada es la determinada por la PC.
- **PWL Cuantizado:** es el modelo PWL Ideal, pero incluyendo los efectos de la digitalización, el truncamiento y las operaciones de punto fijo con el fin de simular el comportamiento ASIC.
- **ASIC:** son los resultados obtenidos directamente por el ASIC, utilizando la interfase de la PC.

Ya definidos los modelos, se presentan entonces, a continuación los tres ejemplos donde se analiza para cada uno los errores y el desempeño de la estructura NOE representada en el ASIC.

Ejemplo 1

En este ejemplo, se considera un conocido sistema dinámico de logística no-lineal bidimensional discreto. La ecuación es la siguiente:

$$y_{i+1} = 0{,}4y_i(1 - y_i) + 0{,}5u_i \tag{3.9}$$

La Figura 3.9 muestra el modelo de la estructura NOE donde se observa una señal de entrada y la realimentación de una sola muestra de la salida. La comparación del sistema real con el modelo denominado ASIC se muestra en la Fig. 3.10.

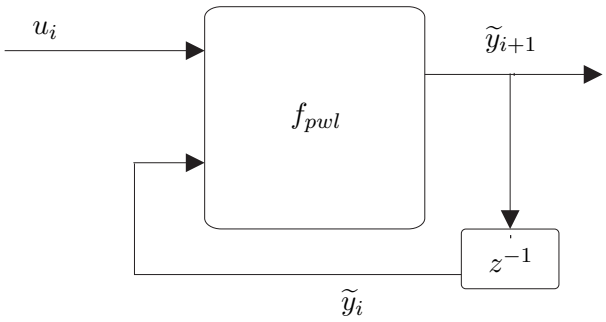

Figura 3.9: Modelo de la estructura NOE con una señal realimentada perteneciente al ejemplo 1. Se observa una unidad retraso.

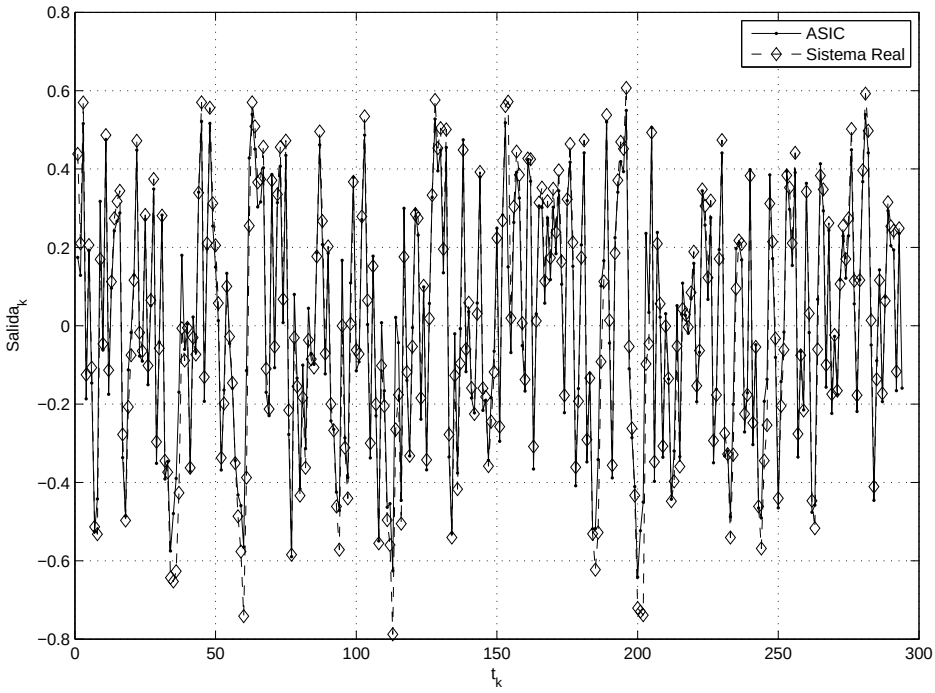

Figura 3.10: Resultados de los 2 modelos: el representado en el ASIC y Sistema Real. Perteneciente a la ecuación del ejemplo 1

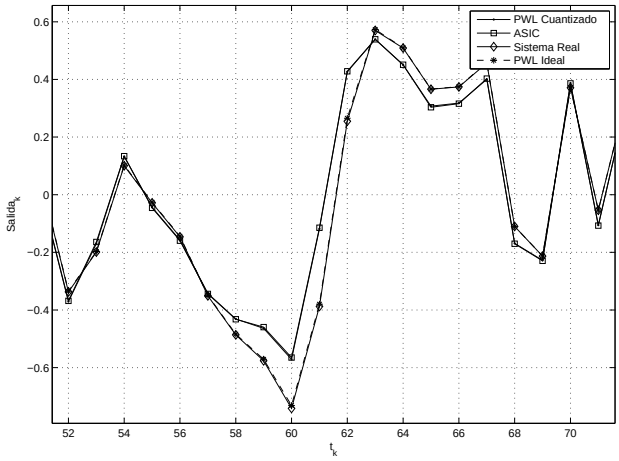

Figura 3.11: Vista ampliada de los 4 modelos: ASIC, PWL Ideal, PWL Cuantizado y Sistema Real. El lugar es donde el error entre los modelos queda más en evidencia. Esto pertenece al ejemplo 1

Cuadro 3.2:
Error cuadrático medio entre el sistema real y los 3 modelos. Perteneciente al ejemplo 1

Sistema Real Vs:	Error Cuadrático Medio
PWL Ideal	0.0015
ASIC	0.0749
PWL Cuantizado	0.0696

Una ampliación de la comparación de los 4 modelos de muestra en la Fig. 3.11. Dicha ampliación se realizó donde la diferencias entre el modelo ASIC y el modelo Sistema Real era mayor para poder distinguir claramente las diferencias.

El cuadro 3.2 muestra los errores ECM para cada uno de los modelos comparados con el Sistema Real.

A modo de ilustración del comportamiento de los modelos, la Fig. 3.12 muestra 3 diferentes curvas de error resultado de las comparaciones entre: el Sistema Real y PWL Ideal; entre ASIC y el PWL Cuantizado y el error entre el ASIC y el Sistema Real. [1] Se puede ver como el PWL Cuantizado se acerca al ASIC y como el PWL Ideal se aproxima al Sistema Real. Las diferencias son consecuencia de la cuantización y truncamiento.

[1] El comportamiento del error es similar en los 3 ejemplos pero solo se muestra para el primer ejemplo.

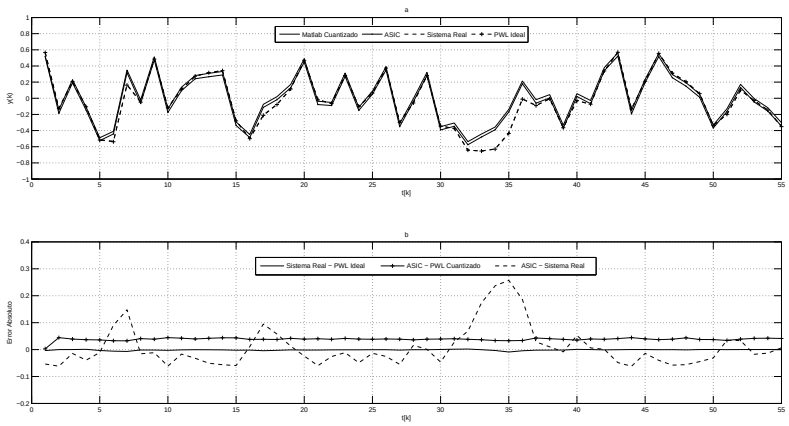

Figura 3.12: a) Resultado de los 4 modelos: ASIC, Sistema Real, PWL Cuantizado y PWL Ideal. b) Error absoluto entre ASIC y el Sistema Real, elASIC y el PWL Cuantizado y por último el Sistema Real y el PWL Ideal. Se puede ver como los procesos de cuantización y truncamiento produce las diferencias entre los modelos. Estas curvas pertenecen al ejemplo 1.

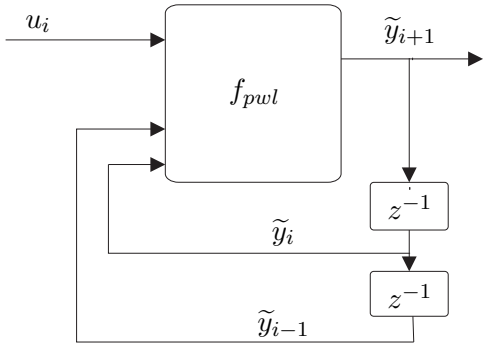

Figura 3.13: Modelo de la estructura NOE con dos señales realimentadas. Perteneciente al ejemplo 2.

Ejemplo 2

En este ejemplo se considera un sistema no-lineal discreto de 3 dimensiones. La ecuación es la siguiente:

$$y_{i+1} = \frac{(y_i y_{i-1})}{(1 + y_i^2 + y_{i-1}^2)} + u_i \tag{3.10}$$

La Figura 3.13 muestra el modelo NOE. Se aprecia una entrada y la realimentación de dos muestras de la salida.

La Figura 3.14 muestra el resultado de los 4 modelos aplicado al ejemplo número 2. Solamente se muestra una ampliación del resultado final de manera de poder observar el rendimiento de los modelos.

El cuadro 3.3 muestra los errores cuadráticos medio para los modelos comparados con el Sistema Real.

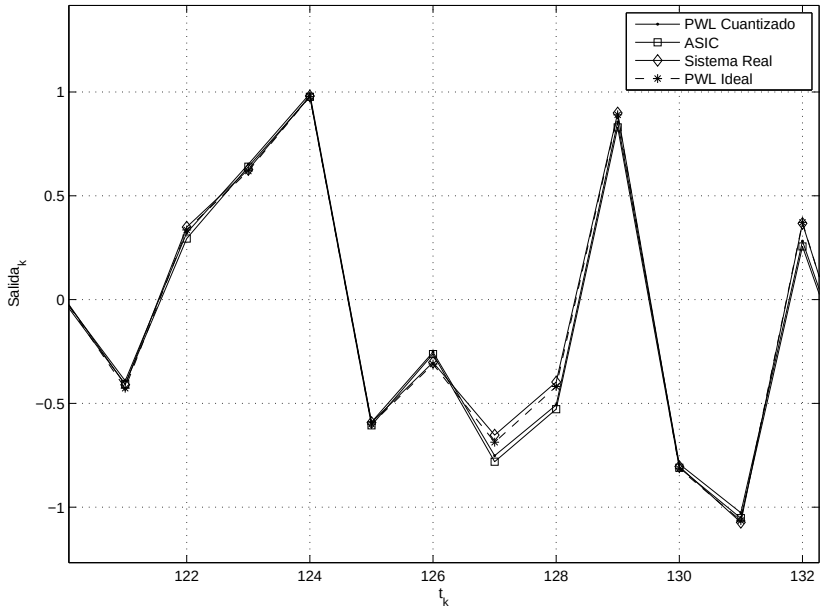

Figura 3.14: Vista ampliada de los 4 modelos: ASIC, PWL Ideal, PWL Cuantizado y Sistema Real. Perteneciente al ejemplo 2.

Cuadro 3.3:
Error cuadrático medio entre el sistema real y los modelos. Perteneciente al ejemplo 2.

Sistema Real Vs:	Error cuadrático medio
ASIC	0.0625
PWL Cuantizado	0.0576

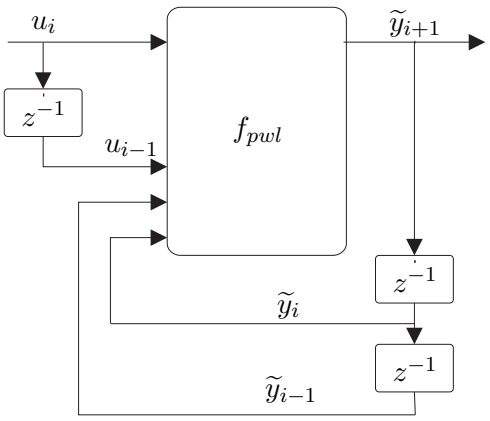

Figura 3.15: Modelo de la estructura NOE con dos señales realimentadas y una entrada retrasada. Estructura perteneciente al ejemplo 3.

Cuadro 3.4:
Error cuadrático medio entre el sistema real y los modelos. Perteneciente al ejemplo 3.

Sistema Real Vs:	Error cuadrático medio
ASIC	0.1927
PWL Cuantizado	0.1894

Ejemplo 3

En este ejemplo, se considera un sistema de 4 dimensiones de manera de poder evaluar el desempeño del chip para un sistema de mayor complejidad. La ecuación es la siguiente:

$$y_{i+1} = -0{,}7y_i^2 - 0{,}7y_{i-1}^3 + 0{,}6u_i + 0{,}6u_{i-1}^2 \tag{3.11}$$

La Figura 3.15 muestra el modelo de la estructura NOE. El modelo incluye una entrada, un retraso de la entrada anterior y la realimentación de dos salidas retrasadas.

La Figura 3.16 exhibe una vista ampliada mostrando el resultado de los 4 modelos aplicados al ejemplo 3.

En el cuadro 3.4 se observan los errores cuadráticos medios para los modelos comparados con el Sistema Real.

En este ejemplo, el tamaño de la memoria se limitó a 4K Bytes, por lo tanto la cantidad de divisiones se limitó a 7 en vez de 15 como en los anteriores. Este cambio de divisiones modificó el rango de los parámetros de memoria debido al truncamiento, como se explicó anteriormente, por lo tanto el nuevo rango es [0 0x6F] y esto afecta considerablemente el error introducido por esta cuantización. Esto se puede ver en el valor alto del ECM, en comparación con los 2 ejemplos anteriores.

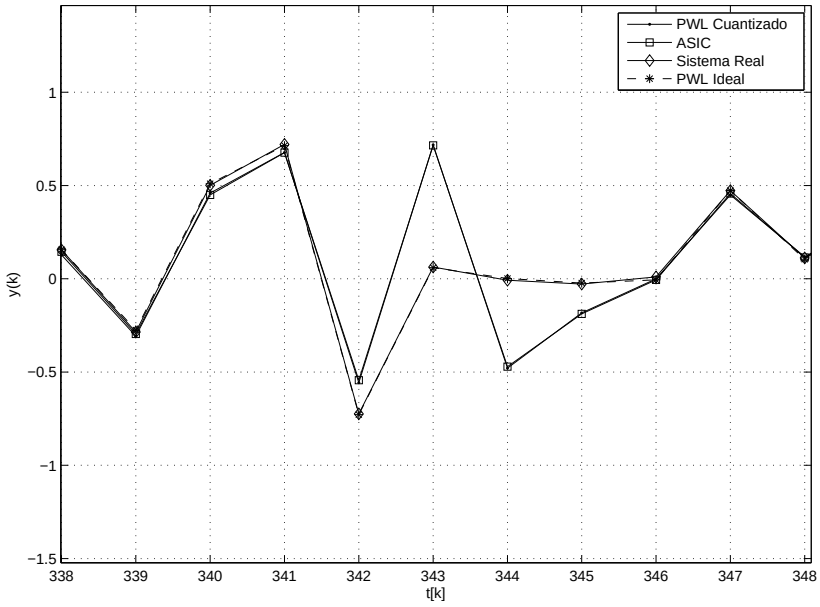

Figura 3.16: Vista más refinada de los 4 modelos: ASIC, PWL Ideal, PWL Cuantizado y Sistema Real. La diferencia se atribuye a la disminución de la cantidad de bits de cuantización de los parámetros de memoria para adaptar el rango de entrada a las señales realimentadas. Perteneciente al ejemplo 3.

Implementación digital del proceso de identificación

A continuación se muestra un ejemplo de identificación digitalizado e implementado en FPGA utilizando datos de entrada-salida de un sistema simulado. El sistema elegido para identificar es el representado por la Ec. 3.9 de los ejemplos anteriores. La señal de entrada al sistema se realizó en forma aleatoria. Ambas señales, entrada y salida, se muestran en la Fig. 3.17

Es muy común en estas operaciones definir dos etapas: primero una etapa de entrenamiento con un conjunto de los datos, y luego una etapa de validación, con otro conjunto de datos. La definición sería la siguiente:

- Entrenamiento: Proceso de ajuste de los parámetros donde se aplica el algoritmo de identificación propiamente dicho. Los datos de entrada - salida de esta parte son los datos de entrenamiento y son los que se usa para actualizar los parámetros de memoria de manera que el modelo NOE responda de igual manera que el sistema caja negra.

- Validación: Una vez finalizado el entrenamiento es necesario ver cómo el modelo generaliza el sistema para otro conjunto de datos distinto al del entrenamiento. Esta claro que un buen resultado de la etapa de validación o generalización está estrechamente ligada a un buen entrenamiento.

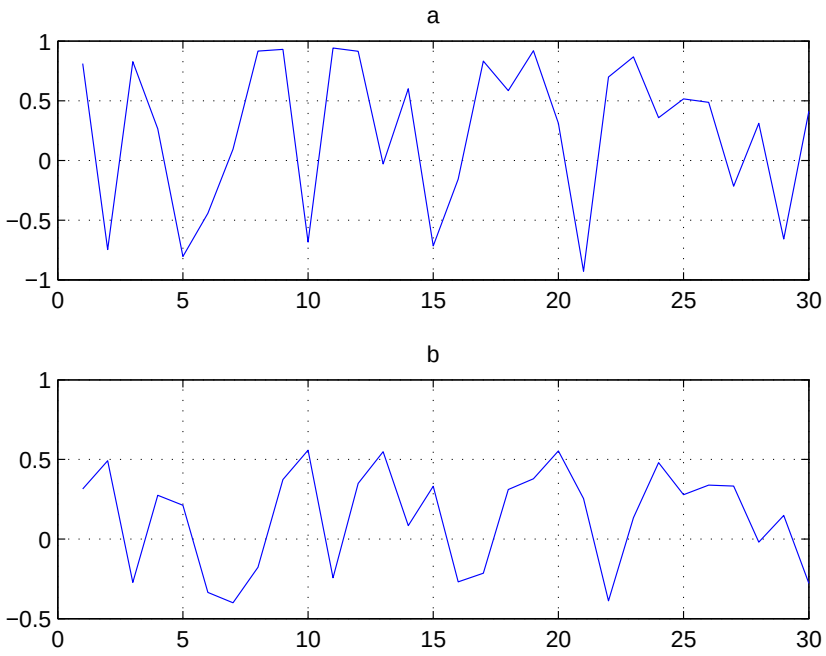

Figura 3.17: a) Señal de entrada al modelo representado en Matlab. b) Respuesta del modelo analítico perteneciente al ejemplo 1 - Ec. 3.9)

Identificación y validación realizado en Matlab

Para poder evaluar el desempeño de los procesos de identificación y de validación de un sistema digital, primero se tiene que crear el modelo del sistema y el modelo NOE de referencia. Por lo tanto, el primer paso a realizar es utilizar un modelo NOE en Matlab y el sistema elegido es el mostrado en el ejemplo 1 (Ec. 3.9). En estos modelos la precisión está definida por la PC.

Una forma útil de observar el desempeño de la etapa de entrenamiento es ver el error que se obtiene entre el sistema real y el modelo NOE para cada iteración que se realiza.

Este error es el ECM entre el sistema real y la respuesta del modelo NOE. La Fig. 3.18 da cuenta de la velocidad de convergencia y permite evaluar si hay que hacer algún cambio en los parámetros del regresor para mejorar el rendimiento. Se puede ver que a medida que se realizan las iteraciones el error disminuye como consecuencia que el modelo NOE va capturando el comportamiento del sistema real.

Una vez finalizado el entrenamiento, se procede con la validación o generalización mostrado en la Fig. 3.19. La validación se realizó con un conjunto de 500 puntos de entrada - salida. La Fig. 3.20 es una ampliación para ver más en detalle la validación.

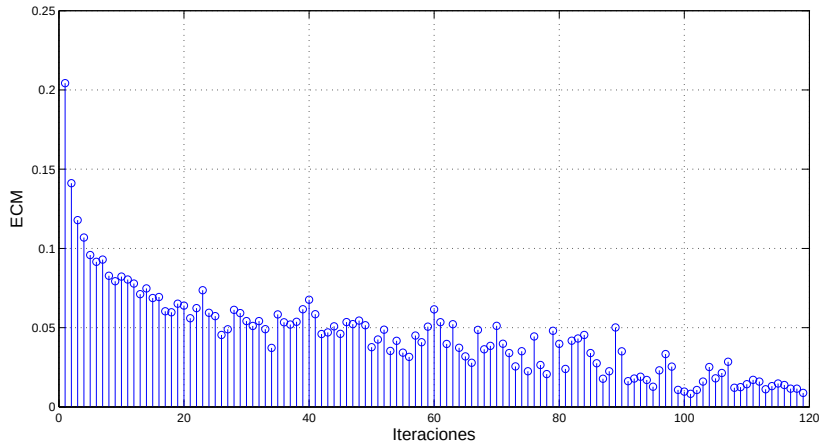

Figura 3.18: ECM de cada iteración para un entrenamiento de 3k puntos de entada salida. Se observa como el ECM va disminuyendo. (Aplicado al ejemplo 1)

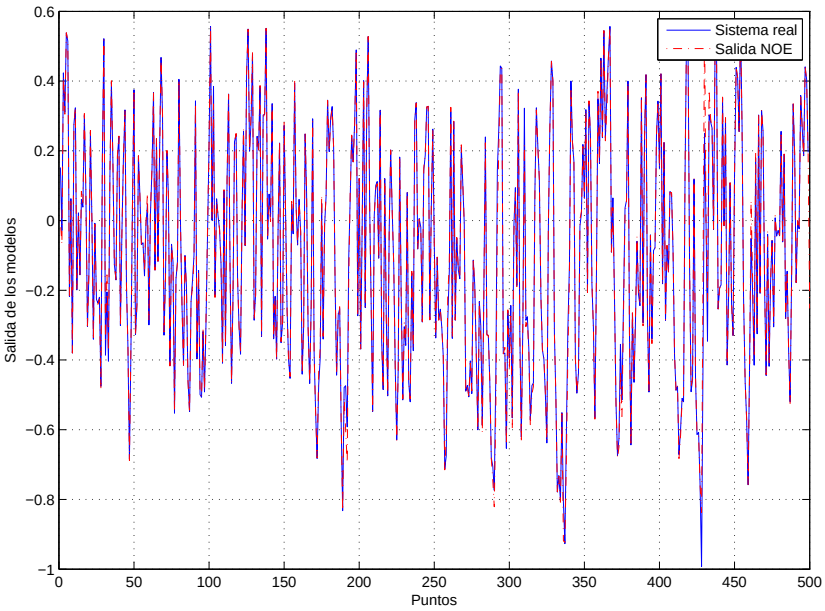

Figura 3.19: Resultado de la validación del modelo NOE implementada en Matlab donde se usa una librería propietaria de PWL.

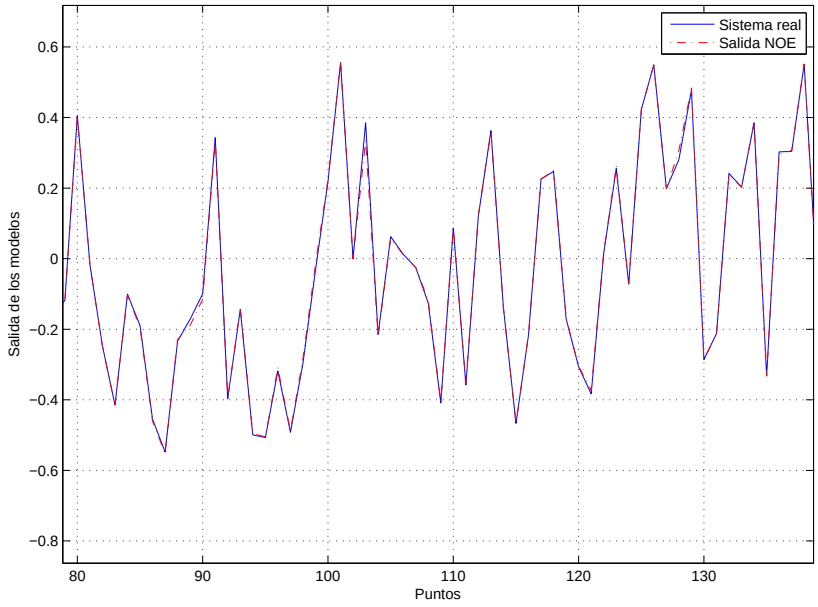

Figura 3.20: Ampliación del resultado de la validación del modelo NOE para ver más en detalle.

Identificación y validación realizado en la FPGA

Una vez obtenidos los datos del modelo en Matlab, se cuantizan y se bajan a una FPGA de manera de poder realizar los procesos de identificación y validación en una estructura digital. Es en esta implementación digital donde todos los efectos de cuantización y aritmética de punto fijo están presentes.

En primer lugar se muestra, en la Fig. 3.21, como es el error del entrenamiento para cada iteración realizada. Se puede ver, comparando con la Fig. 3.18, cómo el error presenta una asíntota horizontal, esto se debe a la digitalización y las operaciones de punto fijo. Una vez finalizado el entrenamiento se procede a la validación. El resultado se muestra en la Fig. 3.22 y para más detalle se realiza una ampliación en la Fig. 3.23. La cantidad de puntos que se utilizaron para la validación, fue de 500 puntos de entrada - salida.

Modelo cuantizado en Matlab

Para demostrar el desempeño del sistema de identificación se adaptó el modelo de la estructura NOE realizado en Matlab agregando los efectos de digitalización. Los resultados son mostrados en la Fig. 3.24. Este modelo simula las mismas operaciones y digitalización que se realizó en la FPGA.

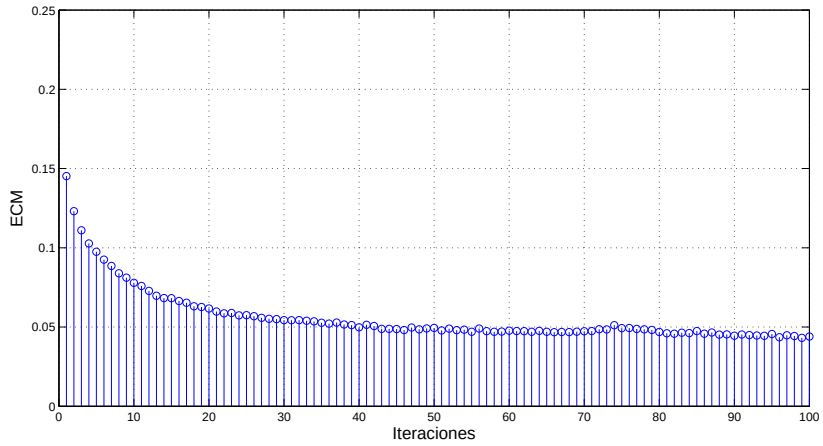

Figura 3.21: ECM de cada iteración para la parte de entrenamiento en el chip. Se ve como el error alcanza una asíntota debido a la digitalización y las operaciones de punto fijo.

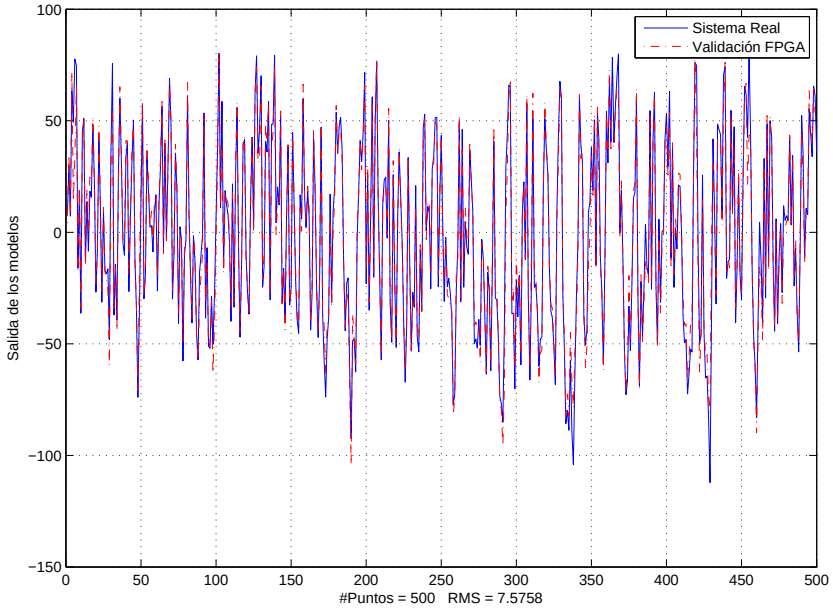

Figura 3.22: Resultado de la validación obtenido por el modelo NOE implementada en la FPGA.

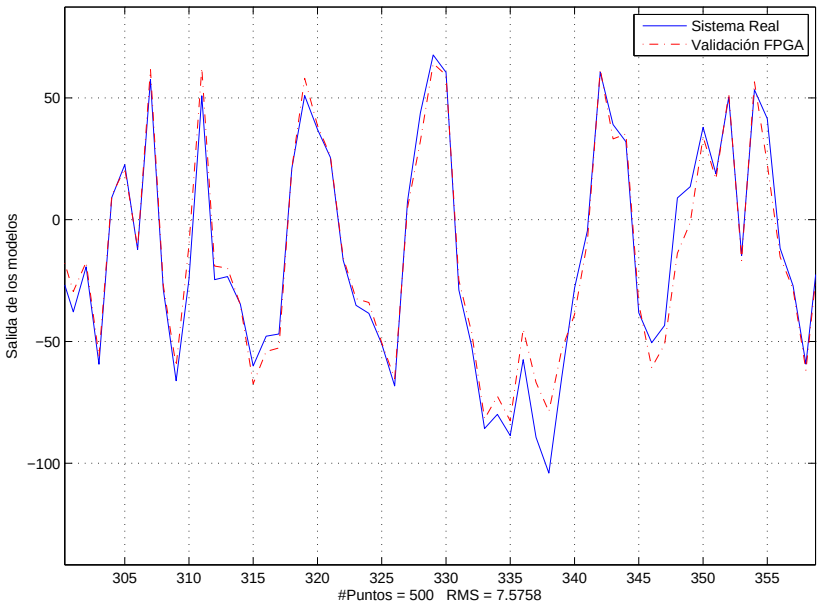

Figura 3.23: Ampliación del resultado de la validación. Se observa las diferencias debido a la digitalización y operaciones punto fijo.

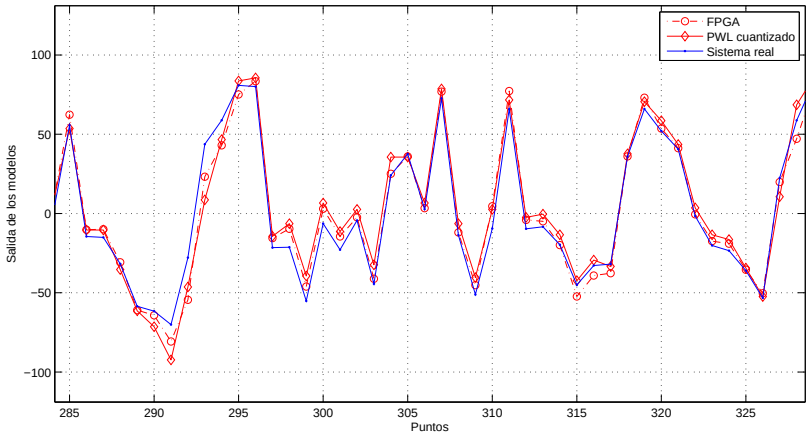

Figura 3.24: Resultados de la validación de los 3 modelos: FPGA, Sistema Real y PWL cuantizado. Se puede ver como la FPGA se ajusto al Matlab cuantizado.

Comentario sobre las velocidades

Un punto importante para señalar es el de los tiempos involucrados en el proceso de identificación para la implementación digital ya que esta implementación podría formar parte de un sistema más complejo, donde la velocidad con la cual se obtiene el modelado de una planta es un factor importante en el desempeño de un controlador. (Un posible lazo de control se muestra en el Apéndice B).

El algoritmo de identificación se ejecutó en 762ms considerando 3000 datos de entrada - salida y con 100 iteraciones. La tasa efectiva de datos del modelo NOE fue de 446 KHz debido a la latencia del sistema y su propia realimentación. La implementación se realizó en una FPGA Virtex 5, cuyo reloj era de 100 MHz. Es necesario mencionar que el PWL utilizado no esta optimizado para máximo rendimiento, de hecho su versatilidad opaca su óptimo rendimiento.

3.5. Conclusiones

En este capítulo se trabajó en la utilización de una función PWL para obtener una estructura NOE.

Se evaluó el desempeño de la implementación digital de una estructura PWL y los errores que se introdujeron por la cuantización y aritmética de punto fijo. Para ello, se implementaron sistemas con distintas complejidades en cuanto a la cantidad de variables que se incluían en la función PWL.

Se estudiaron las alternativas existentes para la obtención de los datos de un sistema de caja negra: On-line y Batch, y se plantearon dos formas de aplicar el algoritmo de identificación: Acumulativa e Instantánea. Se enumeraron las ventajas y desventajas de estas formas de manera de indicar su conveniencia ante un problema existente donde se requiere implementar una solución.

Se presentó un implementacjón del algoritmo instantáneo desarrollado en una FPGA poniendo de manifiesto los errores causados por: las operaciones con punto fijo y cuantización. Además se presento la arquitectura de los bloques del algoritmo y se obtuvieron las velocidades alcanzadas de manera de tener un parámetro para sistemas de control más complejos.

Capítulo 4

Estructuras PWL-FIR

4.1. Introducción

En este capítulo se desarrolla la corrección de un ADC utilizando una estructura PWL. La motivación se basa en la demanda de dispositivos ADC de muy de alta velocidad y baja distorsión, como consecuencia de los sistemas digitales complejos. En general, los ADC de alta velocidad presentan efectos no lineales en su función de transferencia que causan distorsión en la señal de salida discreta [50]. Estas no linealidades se originan en diferentes subsistemas del convertidor, y afectan a la salida determinando un deterioro severo en el rendimiento general. El resultado es la presencia de la distorsión armónica que degrada el rango dinámico en la banda de frecuencia de interés. Por lo tanto, la compensación de no linealidad es necesaria.

En la cita [51], se presenta una breve descripción de las principales estrategias de compensación que se encuentran en la bibliografía especifica. Estas incluyen métodos de búsqueda basados en tablas, desambiguación y métodos basados en un modelo de inversión. Los primeros son bien conocidos y han demostrado ser eficaces para la corrección de errores estáticos, pero son costosos en cuanto a requerimientos de memoria cuando se considera incluir la corrección de errores dinámicos. Además, carecen de las propiedades de generalización. Es decir sólo son capaces de corregir los errores que han sido previamente caracterizados y almacenados [52].

Por su parte, los métodos de desambiguación consisten, en general, en la aplicación de una señal de ruido pseudo-aleatorio en la entrada del ADC para descorrelacionar el ruido de cuantización de la señal de entrada, pero el comportamiento no lineal no es captado por esta solución y por lo tanto hay errores que no se pueden compensar [50, 51]. Por lo tanto, el modelo de inversión y de post-compensación son opciones atractivas y esto es motivo de investigación reciente.

Otro enfoque es el de una compensación externa, donde no se necesita información detallada sobre la física que rige las no linealidades en la salida [53]. Su comportamiento se mide y se caracteriza y luego un circuito externo compensa el error cometido. La desventaja es que la

corrección ocurre en el punto de operación donde se midió, es decir para un rango dinámico y frecuencia de la señal de entrada, desmereciendo su aplicación.

Una posible aplicación para la corrección es el uso de técnicas de modelos de post-compensación digital para reducir la distorsión. Estas técnicas se basan generalmente en la aplicación de una distorsión adicional a la salida digital del ADC, que cancela las distorsiones originales presentes en la salida del dispositivo [54], [55]. Estas técnicas implican en general dos pasos. En primer lugar, el post-compensador tiene que ser entrenado (off-line), utilizando datos de medición del dispositivo. En segundo lugar, se implementa en línea a la salida del ADC para ir corrigiendo los datos de salida en tiempo real.

El método elegido en este trabajo consistió en la post-compensación donde la motivación inmediata fue aplicar una estructura PWL para la corrección de un ADC. Básicamente se propone identificar mediante el modelo PWL el error que se produce en la transferencia lo que hace que se degrade el desempeño del dispositivo. Conocido es por la literatura que se trata de una transferencia multivariable no lineal no conocida. La misma es multivariable ya que el error cometido en la muestra observada de salida se relaciona con las muestras de entradas anteriores. La forma que se estableció para poder medir el desempeño de la corrección fue evaluar el número efectivo de bits que se obtienen a la salida del dispositivo.

Se plantea en este capítulo la utilización de una estructura PWL para implementar un modelo FIR. Se presentan las ecuaciones para poder modelar el error del dispositivo bajo estudio, para luego, mostrar el algoritmo de identificación utilizado. Dado que el trabajo se realizó sobre un dispositivo comercial, se presentan sus características y los parámetros con que se califica a un dispositivo ADC. Se distinguen los tipos de señales que se usaron para la identificación experimental, se presentan los resultados del laboratorio y las conclusiones que se obtuvieron para este capítulo.

4.2. Metodología de corrección

En esta sección se plantea como utilizar una función FIR implementada con una estructura PWL para la corrección de un dispositivo ADC. Recordando la expresión analítica de un filtro FIR discreto, 2.3, se tiene lo siguiente:

$$\begin{aligned} y[n] &= b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + \ldots + b_m x[n-m] \\ y[n] &= \sum_{i=0}^{m} b_i x[n-i] \end{aligned} \tag{4.1}$$

donde m es la cantidad de muestras pasadas de la entrada que se toman para el cálculo de la función. Las componentes b_i son los coeficientes del filtro que tienen que ser calculados para una cierta función de transferencia requerida. La Ec. 4.1 determina que la salida en el instante n es la suma ponderada de las muestras pasadas. Además, considera que entre las muestras pasadas no existe relación de ningún tipo.

Aplicando esta misma idea en una función PWL se logran obtener combinaciones de las muestras retrasadas de cualquier índole, expresándolo de la siguiente manera:

$$\begin{aligned} y[n] &= f(x[n], x[n-1], x[n-2], \ldots, x[n-m]) \\ y[n] &= \lambda(x[n], x[n-1]) + \zeta(x[n-2], x[n-5]) + \varphi(x[n-4], x[n-m]) \end{aligned} \tag{4.2}$$

donde λ, ζ y φ son funciones cualesquiera. Esto es debido a que la transferencia podría ser cualquiera de las combinaciones de muestras de la entrada atrasadas en el tiempo. Es decir, por ejemplo, que la transferencia puede ser función solamente de $n-2$ y $n-5$.

La idea radica en identificar estas funciones λ, ζ y φ cuando no se conoce la transferencia de forma analítica del sistema en estudio. Por lo tanto pasa a ser un problema que se conoce como identificación tipo caja negra, donde solamente se conocen la entrada y la salida. Por ende, no se conoce con anticipación que entradas anteriores influyen en la salida actual.

La metodología que se empleó es la de post-compensador. Es decir, no se conoce la estructura interna del dispositivo ADC. La idea básica consiste en comparar la salida del ADC real con la salida de un dispositivo ADC ideal (realizado en Matlab). De esta comparación se obtiene un error y este error es el que se desea reproducir o estimar con la estructura PWL. Obtenido el error, éste se suma a la salida original y esto permite mejorar el desempeño final del dispositivo ADC.

Una vez definida la estructura FIR, se procede a definir la función error, para esto hace falta mencionar dos aspectos de la señal para poder realizar la comparación entre los dos dispositivos ADC, real e ideal.

Fase de la señal de entrada

El primer paso para establecer el error es estimar la fase de la señal de entrada al ADC real respecto de la fase de la señal de entrada al ADC ideal realizado en Matlab. Esta fase se obtiene correlacionando ambas salidas entre si y buscando el mínimo error entre ellas. De esta manera se obtiene la menor diferencia de fase entre el ADC real e ideal. La Figura 4.1 muestra el proceso para obtener el error entre los dos ADC: ideal y real, ilustrando el bloque de puesta en fase.

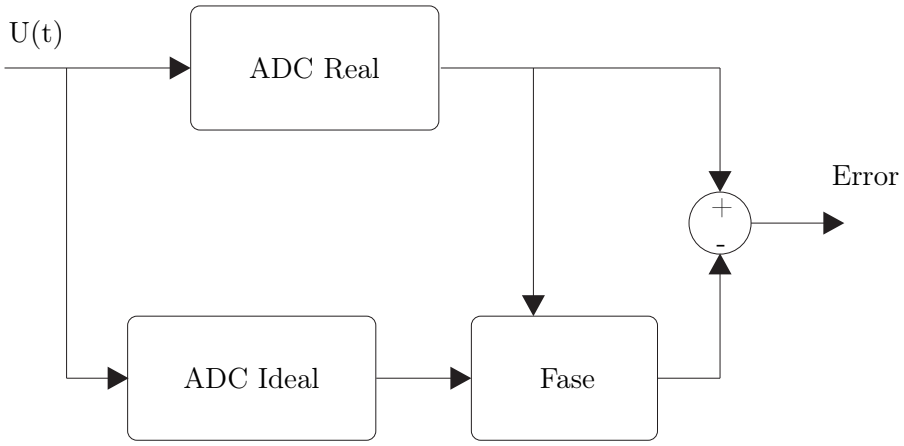

Figura 4.1: La señal de entrada es procesada por el ADC ideal y la salida es puesta en fase con las muestras provenientes del ADC real. El error se obtiene como la diferencia de ambos ADC.

Armónicos de la señal de entrada

Una cuestión importante es que la señal de entrada se puede tomar como una señal ideal que consiste en un tono puro o se pueden incluir los armónicos que fueron identificados en el analizador de espectros. Es decir, para calcular el error, la señal de entrada de referencia se puede tomar con o sin estos armónicos. Se verá más adelante que esto impone un limite a la corrección ya que dichos armónicos influyen en el valor de Signal to Noise and Distortion ratio (SINAD).

Errores de un dispositivo ADC

Es conocido en la literatura que los errores que se producen en un dispositivo ADC se pueden dividir en dos tipos:

- Estáticos: Estos afectan el desempeño del ADC para señales continuas o de baja frecuencias, y afectan, además, significativamente la pureza espectral de la salida, en particular los que resultan en la separación no uniforme de los umbrales de cuantización.

- Dinámicos: Estos incluyen la: distorsión armónica en el muestreo y retención del amplificador, variación de la frecuencia dependiente de la señal de entrada, valores internos de las etapas de amplificación variables con las muestras, y variación de los umbrales de cuantificación dependiente de las muestras de entrada.

Por lo tanto son los errores dinámicos los que sustentan la hipótesis básica de poder estimar el error producido en la transferencia de un dispositivo ADC utilizando una estructura PWL. El sistema de ecuaciones que involucra la función PWL está representado de la siguiente forma:

$$
\begin{aligned}
&f_{pwl}(x_n, \ldots, x_2, x_1) = e \\
&f_{pwl}(x_{k-n}, \ldots, x_{k-1}, x_k) = e_k \\
&f_{pwl}(x_{k-n}, \ldots, x_{k-1}, x_k) = ADC_{Ideal}(k) - ADC_{Real}(k)
\end{aligned} \tag{4.3}
$$

Donde la función f_{pwl} captura la dinámica del dispositivo ADC y n es la cantidad de muestras pasadas. La cantidad de muestras n representa la dinámica del ADC y la cantidad de divisiones representa el tipo o grado de no-linealidad que presenta la transferencia del ADC.

Identificación de los parámetros

Una vez obtenido el error, se procede a la obtención de los parámetros de la estructura PWL. Para esto se definen dos tipos de identificación:

- Método del gradiente: Es una algoritmo similar a la identificación NOE, pero a diferencia de este, las entradas a la estructura PWL son la entrada última y muestras retrasadas de esta. La cantidad de divisiones es variable. Este algoritmo ajusta los parámetros de memoria utilizando el error instantáneo producido, en sentido contrario al gradiente.
- Aproximación por mínimos cuadrados: Este método plantea las ecuaciones lineales y encuentra la matriz solución del sistema. Esta matriz contiene los parámetros de la función PWL que mejor se ajusta al sistema.

El método del gradiente es simular al utilizado en la estructura NOE. La ecuaciones del método por mínimos cuadrados son las siguientes:

$$
c\Lambda = error \tag{4.4}
$$

Donde Λ son los vectores de la base estándar, c son los parámetros que queremos estimar y el error es la diferencia entre los dispositivos ADC real y el ADC ideal. La estimación del vector C se obtiene haciendo la Pseudo inversa de Λ

$$
c = inv(\Lambda^T.\Lambda)\Lambda^T error \tag{4.5}
$$

Se verá más adelante que este método involucra el uso de matrices que pueden llegar a tamaños considerables, ej 32K x 32K, por ese motivo la identificación se hace fuera de línea ya que implica una cantidad de cálculo importante. Una alternativa que se aplicó para reducir la cantidad de puntos fue mejorar la distribución de los datos existentes de manera de manejar la densidad de datos por símplice. Para esto se utilizó una rutina en Matlab que reducía la cantidad de puntos teniendo en cuenta la distancia entre ellos. Básicamente, filtraba los puntos cuya distancia era menor que un valor dado. Es claro que para el proceso de identificación, la

Cuadro 4.1:
Características del dispositivo comercial ADC elegido.

Ítem	Valor
Salida	16 bits (LVDS o CMOS)
Frecuencia muestreo	130 Mega-Samples Per Second (MSPS)
rango de entrada	1,5 V
Máximo de Integral Nonlinearity (INL)	± 5 Low Significant Bit (LSB)

generación de datos de entrada es clave para el éxito del entrenamiento y su posterior proceso de validación o generalización.

4.3. Entorno experimental

El primer paso para la descripción del entorno experimental es describir el dispositivo comercial y las placas de desarrollo donde éste se encuentra montado.

El dispositivo ADC elegido es el AD9461, cuyas características técnicas más importantes son enumeradas en la Tabla 4.1. El último parámetro implica una resolución efectiva de 11 bits.

Este dispositivo viene montado en una placa de desarrollo e incluye una placa adquisidora (Fig. 4.2). Esta última cuenta con una First In First Out (FIFO), de 32K muestras, para almacenar los datos de salida del ADC que luego son enviados hacia la PC para su procesamiento posterior (Fig. 4.3). La placa del ADC tiene por función adaptar la señal de entrada referida a tierra a una señal diferencial montada sobre un nivel de continua de manera de ajustar el nivel recomendado para la entrada diferencial del dispositivo ADC. El mismo proceso lo hace para la señal de reloj. Además provee las alimentaciones del dispositivo.

Otro elemento importante incluido en el paquete del dispositivo ADC es un Software (SW) para la caracterización. Este SW se usó para dos actividades: la primera, adquirir los datos de salida del ADC para su procesamiento en la PC y la segunda, para la verificación de los datos obtenidos en Matlab. Es decir, la elaboración obtenida con el Matlab se verificaba con unas rutinas propias incluidas en el SW propietario de *Analog Device*.

El entorno de trabajo, parte mostrado en la Fig. 4.4, incluía:

Dos generadores de funciones, uno vectorial Rhode & Shwartz Modelo SMU200A y otro de funciones HP Modelo 8640B, un analizador de espectros Agilent Modelo E4407B, un LCR Meter Agilent Modelo E4980A y un Vector Network Analizer (VNA) HP 8714B. El RLC meter que se utilizó para la caracterización de los elementos pasivos que componen los filtros, mencionados haca el final de la sección.

Como consecuencia de la cantidad de bits que presenta el ADC, se torna un parámetro crítico

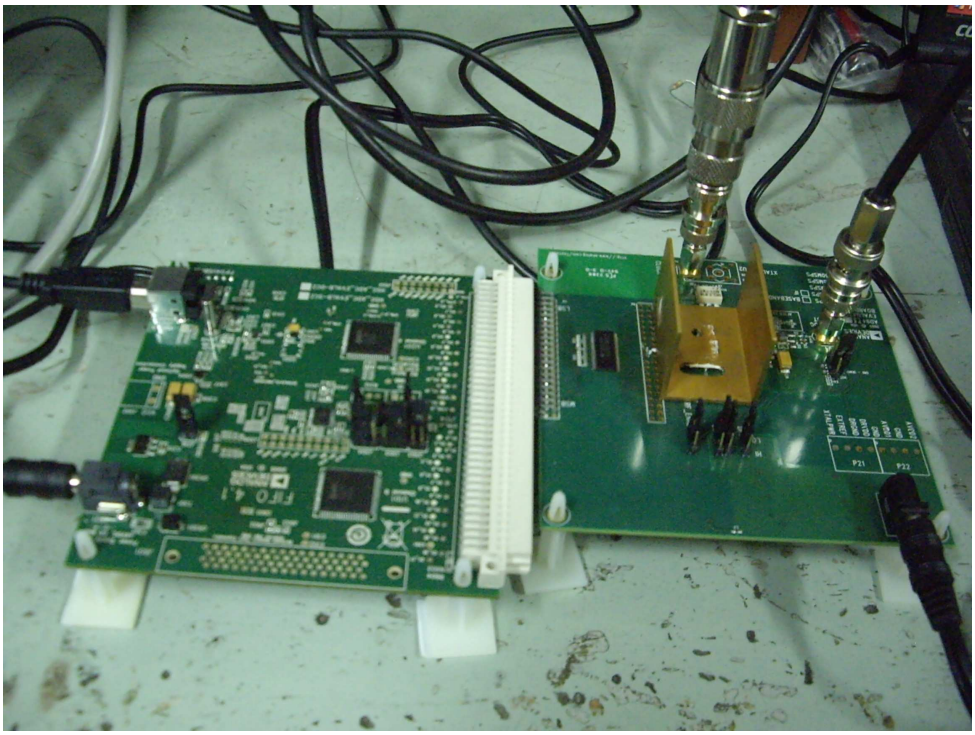

Figura 4.2: Fotos del ADC y de la placa adquisidora del ADC para almacenar los datos provenientes del dispositivo. Estos datos son enviados hacia una computadora para su posterior procesamiento.

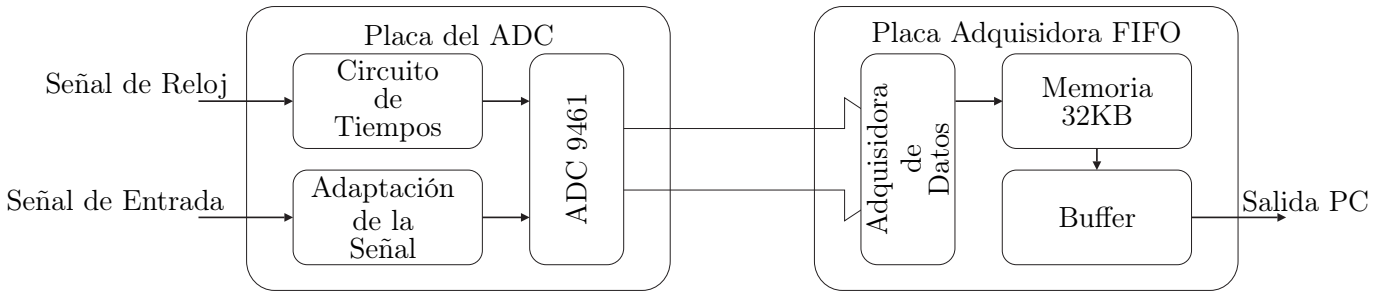

Figura 4.3: Esquema de las placas del ADC y su placa adquisidora.

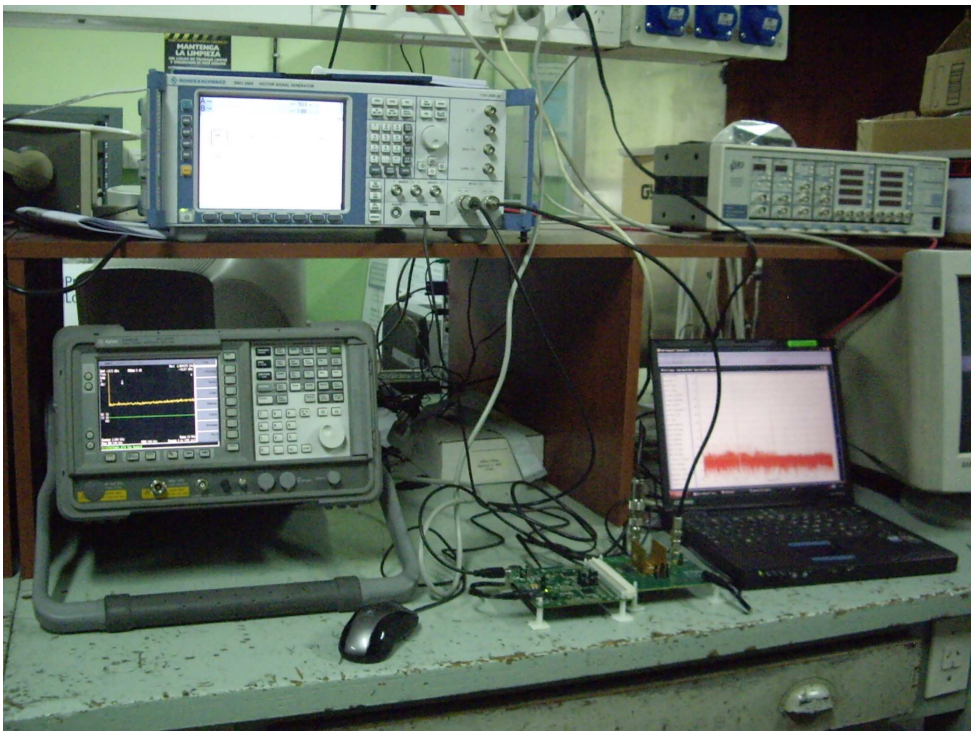

Figura 4.4: Fotos del entorno de trabajo para la caracterización del ADC. Se puede ver el generador de señales y el analizador de espectro.

la pureza espectral del generador ya que ésta, idealmente, tiene que ser de 98 dB.

Por lo tanto se realizó la caracterización del generador $R\&S$ en el laboratorio, ya que éste incluía la opción de señales multitonales. Para la caracterización se uso el valor de SINAD que se obtuvo del SW propietario de Analog Device. La Fig. 4.5 muestra el resultado obtenido para el generador en cuestión. (El generador HP fue caracterizado pero el resultado no se incluye).

Señales de entrada al ADC

Luego de presentado el entorno de trabajo, es necesario analizar qué tipo de señales son necesarias para el entrenamiento o identificación de los parámetros de la estructura PWL.

Una característica importante de los datos de entrada es que puedan aportar la información necesaria para poder estimar los parámetros de la función PWL. De los sistemas lineales se sabe que tienen que existir tantas ecuaciones linealmente independientes como el número de incógnitas. Por lo tanto, para el caso del PWL deberían existir también tantas ecuaciones como el número de parámetros de la función PWL.

Para el caso concreto del ADC, las entradas posibles están limitadas a los generadores disponibles. Estos generadores tienen que cumplir con la potencia necesaria para utilizar todo el rango de entrada del ADC y con la pureza espectral de manera de cumplimentar con la SINAD requerida para el experimento del dispositivo.

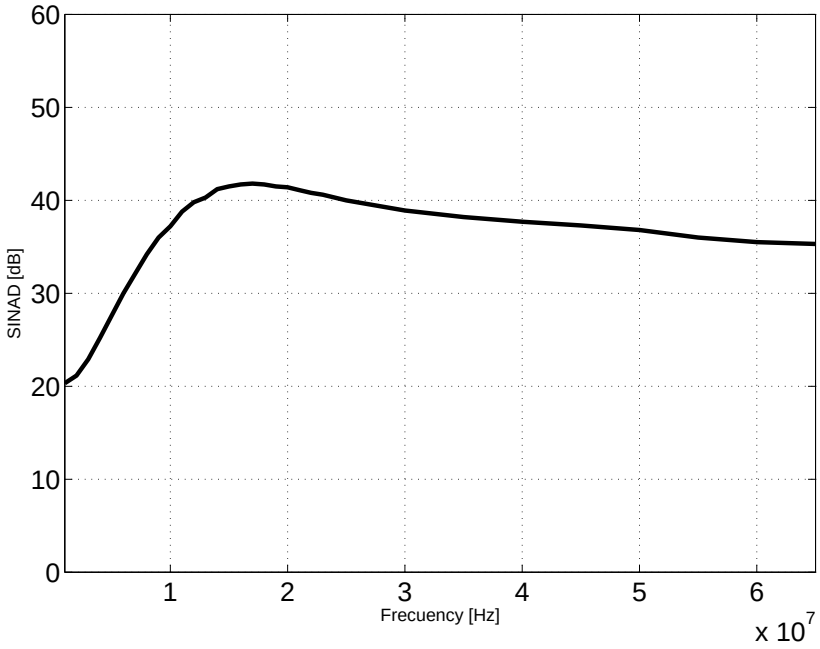

Figura 4.5: SINAD Medido del generador de señales $R\&S$. Se observa que el nivel está muy por debajo del SINAD requerido. Además, se muestra como el rendimiento del generador no es igual en todas las frecuencias.

Con estos dos generadores de señales se compusieron tres secuencias de datos diferentes:

- Tono puro: Es cuando se usa una sinusoidal de una frecuencia dada. Es el más sencillo de los experimentos. Los datos a recolectar son los datos de entrada y la salida del ADC.
- Concatenación de tonos puros: Se arma un vector con distintas muestras de sinusoidales puras. La idea es tomar la misma cantidad de ciclos de diferentes tonos puros y se arma un vector de entrada y salida del ADC. En el futuro, este caso se denomina "FM" (frecuencia modulada) ya que si se gráfica este vector resultante su apariencia es como una señal FM de radio.
- Multitonal: Consiste en tener una frecuencia portadora con bandas laterales formadas por tonos puros de distinta amplitud, frecuencia y fase entre ellos, como muestra la Fig. 4.6

Estos tres conjuntos de datos se usaron para realizar todos los experimentos en el laboratorio y se evaluó la eficiencia de cada uno frente a la generalización obtenida. Es claro que cuanto más parámetros del modelo quedan involucrados para un conjunto de entrada y salida, más rico será el entrenamiento y por lo tanto mejor la generalización. Esta propiedad de utilizar más o menos parámetros se conocen con el término de constelación o modos de los datos de entrada.

A modo de ejemplo, se muestran las constelaciones de los datos de entrada para los tres casos: monotonal, FM y Multitonal. En la Fig. 4.7 se observa cómo es la constelación de datos para

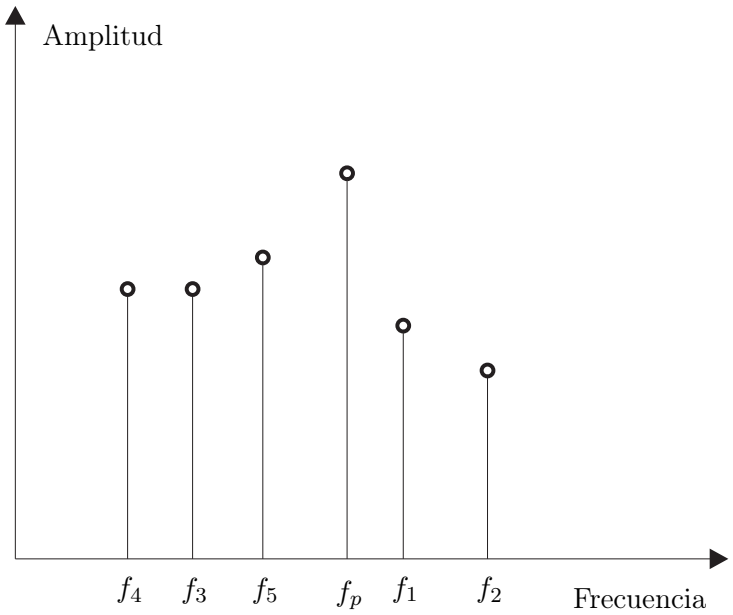

Figura 4.6: Ejemplo de señal multitonal. La F_p es la portadora. La fases de las bandas laterales no esta representada en el gráfico.

un sistema de R2. Esto sería tomando la muestra actual y una muestra retrasada de la señal de entrada. La Fig. 4.8 ilustra la constelación para una FM 32 tonos. La Fig. 4.9 presenta la

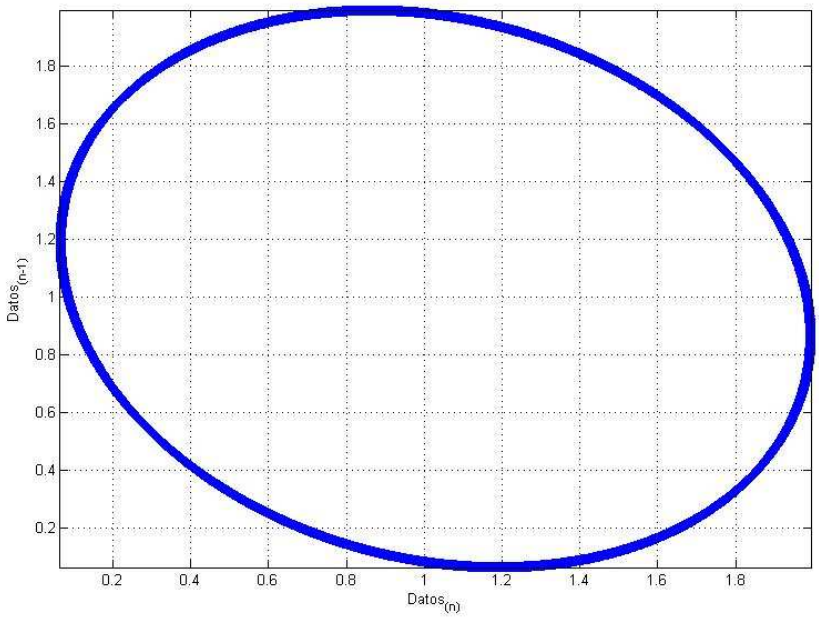

Figura 4.7: Constelación para los datos de entrada de una monotonal para un sistema FIR de R2.

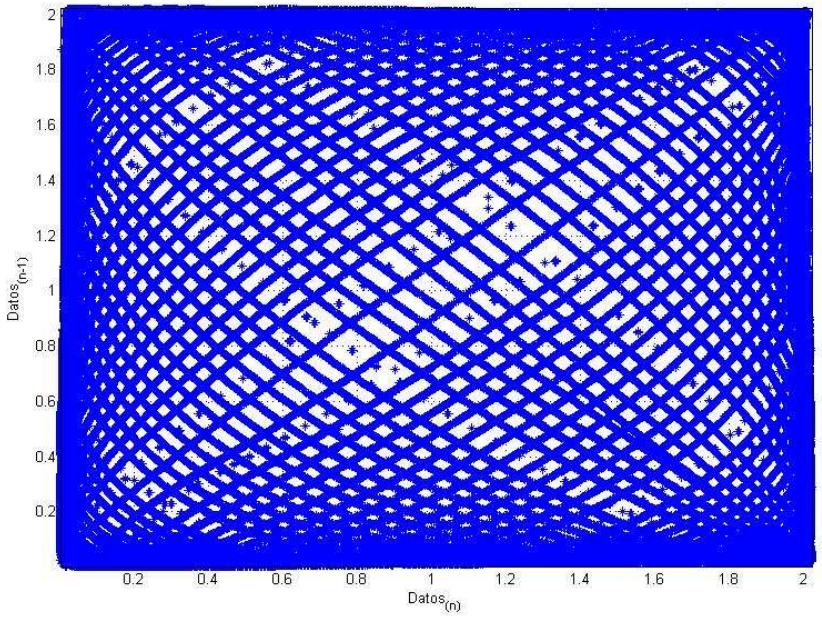

Figura 4.8: Constelación para los datos de entrada de 32 tonos para un sistema FIR de R2.

constelación para una multitonal de 5 tonos.

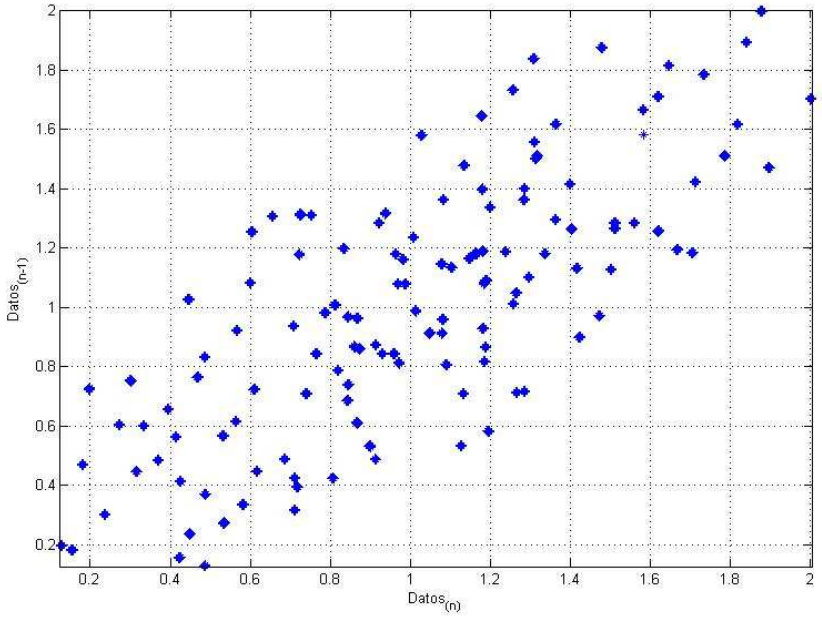

Figura 4.9: Constelación para los datos de entrada de una multitonal de 5 tonos para un sistema FIR de R2.

La Figura 4.10 muestra cómo es la constelación de datos para un FIR de tres dimensiones con una FM de 5 tonos Se puede apreciar de estos ejemplos de constelaciones que el que presenta

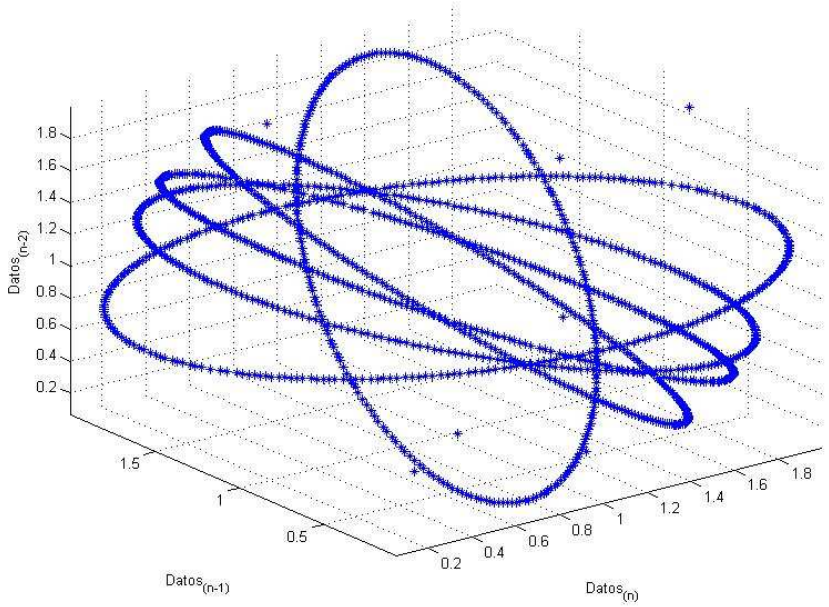

Figura 4.10: Constelación para los datos de entrada de una FM de 5 tonos para un sistema FIR de R3.

mayor dispersión de los datos es la multitonal o la FM de varios tonos, 32 en este caso. Pero como se verá más adelante, esto implica una limitación para el filtrado de las señales.

Un problema general que abarca a todos los tipos de identificación es la generalización del modelo, como se vio en el capítulo anterior. De ahí surge la necesidad de contar con datos de entrenamiento lo suficientemente ricos para que abarquen todos los modos del sistema.

Si bien es muy difícil decir qué señal es la mejor para el entrenamiento, se diseñó un experimento para evaluarlo. El experimento consistía en observar qué parámetros no fueron evaluados durante el proceso de identificación. Es decir, cuál fue el recorrido de señal de entrada en la estructura PWL. Está claro que cuantos menos parámetros fueran evaluados, la generalización final se vería perjudicada.

Para proceder con el experimento se armó una estructura PWL de 4 dimensiones (3 muestras hacia atrás) y se varió el número de divisiones del grillado. Luego, se evaluó el porcentaje de parámetros que permanecieron en 0, es decir que el recorrido de los datos de entrada no lo incluyó. Las Figuras: 4.11, 4.12 y 4.13 muestran el porcentaje de parámetros que quedaron en 0 para distintas entradas. Se puede apreciar cómo rápidamente el porcentaje de parámetros en cero aumenta cuando el número de divisiones lo hace también. Además, se observa que el vector FM es el que mejor generalización presenta, pero su implementación es más compleja por la cantidad de filtros a utilizar como se explica a continuación.

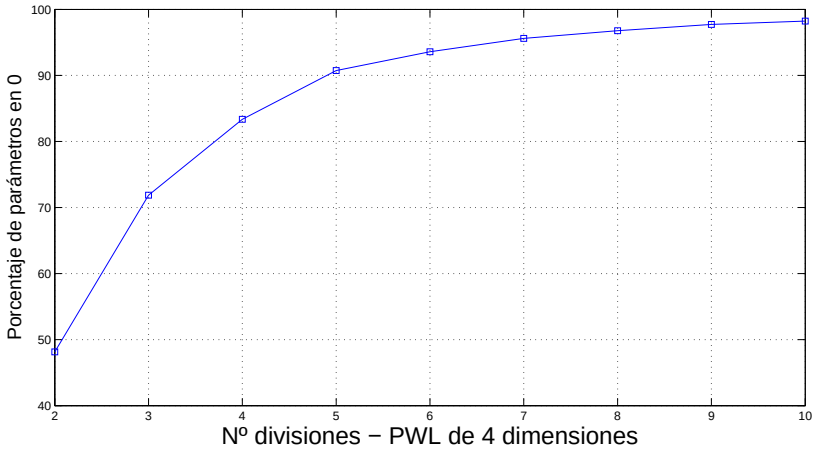

Figura 4.11: Generalización de una monotonal. Se puede ver el porcentaje de parámetros en 0, es decir parámetros que no fueron evaluados por este set de datos.

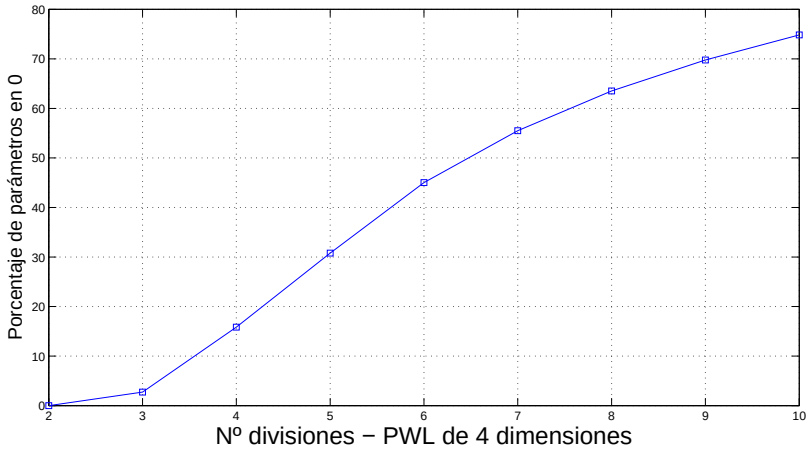

Figura 4.12: Generalización del vector FM. Se distingue el porcentaje de parámetros que no fueron evaluados por este set de datos.

Filtrado de la señal de entrada

Como se explicó anteriormente, es necesario adecuar la señal de entrada para poder trabajar con el dispositivo. A continuación se explica el proceso para mejorar el nivel de SINAD correspondiente para la resolución en bits del dispositivo. La ecuación (4.6), conocida de la literatura, relaciona la pureza de la señal con referencia al ruido en función de la cantidad de bits del ADC.

$$SNR_{dB} = 6{,}02N + 1{,}76 \tag{4.6}$$

Donde N es la cantidad de bits. Por lo tanto se necesita un Signal to Noise Ratio (SNR) de 98 dB, debido a los 16 bits del dispositivo comercial. Es decir, si la señal de entrada tiene menos

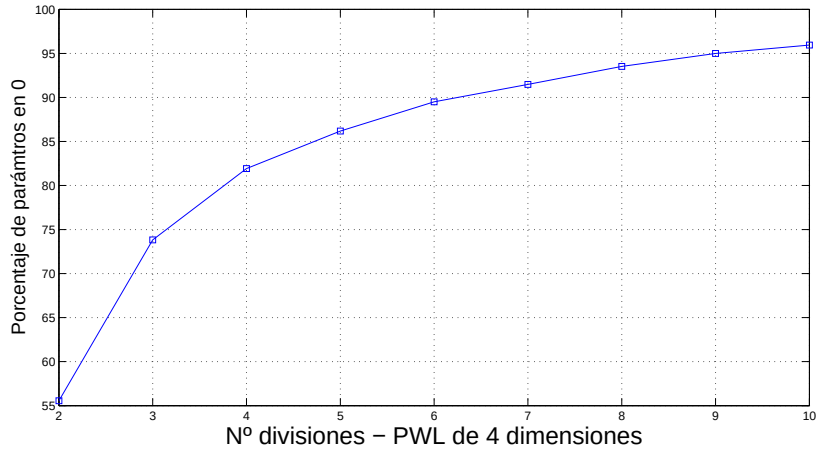

Figura 4.13: Generalización de la señal multitonal producida por el generador. Se observa el porcentaje de parámetros que no fueron evaluados por este set de datos.

de 60 dB, la transferencia del ADC no afecta a la salida ya que según la hoja de datos el peor rendimiento del dispositivo son 10 bits.

Un parámetro que involucra el SNR y la distorsión es el SINAD. Este parámetro incluye el ruido propio del generador y los armónicos que introduce el mismo, por no ser un generador ideal. Además, se observó que estos armónicos son influidos según el rango de potencia que se extrae del generador.

A partir del estudio realizado con el analizador de espectro de los generadores, se eligió el HP 8640B. Este equipo tenía un SINAD de alrededor de 50 dB. Por lo tanto es claro que se debía filtrar la señal de entrada al dispositivo ADC agregando filtros pasivos analógicos. Estos filtros fueron caracterizados con el Analizador Vectorial de Redes. Un ejemplo de los filtros utilizados se ve en la Fig. 4.14 siendo un Butterworth de 6to orden. Las inductancias cuyos valores estaban

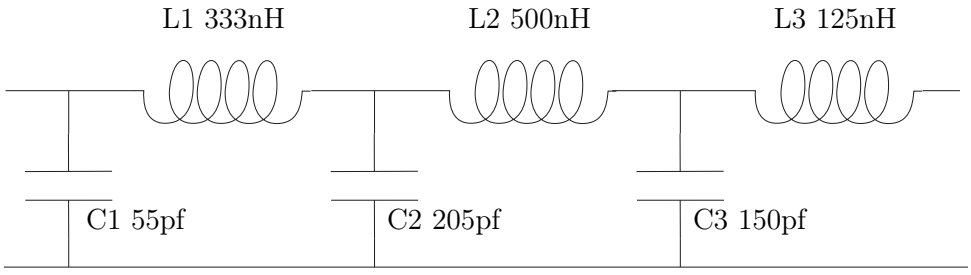

Figura 4.14: Esquema del filtro analógico pasivo, Butterworth de 6to orden, que se implementó para el filtrado de la señal del generador.

dentro de los valores comerciales fueron implementadas con inductancias de choque. Los otros valores fueron realizados en el laboratorio y sus valores fueron medidos con el LCR Meter. Las capacidades fueron adaptadas a combinaciones de capacitores comerciales. El esquema de filtrado

se ilustra en la Fig. 4.15.

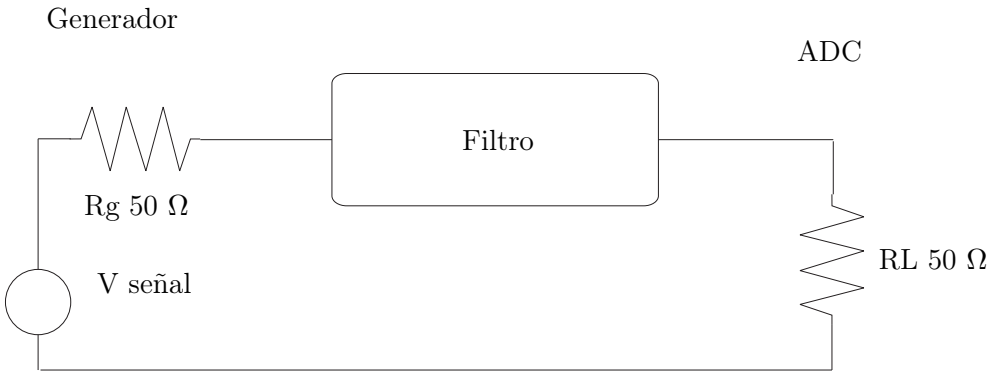

Figura 4.15: Esquema del conexionado del generador, filtro y el ADC.

La decisión de usar filtros pasivos fue para evitar un diseño activo que garantice el SINAD en los niveles que se pretendía. Alternativamente a los filtros pasivos, se probó usando cristales de cuarzo en serie con la señal, pero estos incluían una fuerte atenuación en la frecuencia de paso debido a la poca resolución del generador y esto hacía que los armónicos crezcan, desmereciendo el filtrado.

Los filtros que se usaron se acomodaron en una jaula de Faraday de manera de evitar señales de radios de frecuencia modulada y de televisión que están en el rango de trabajo. Con estos filtros se alcanzó un SINAD de alrededor de 74 dB. Esto implica 2 bits, algo que se puede mejorar, ya que la hoja de datos indica 10 bits para el mejor desempeño del dispositivo.

La aplicación de estos filtros para alcanzar el SINAD requerido imposibilita el uso de señales multitonales ya que exige un filtro analógico más difícil de implementar. Además, se comprobó que el generador que produce este tipo de señales multitonales tiene un SINAD alrededor de los 38 dB resultando en un peor desempeño en comparación al otro generador disponible, lo que limitó a trabajar con monotonales y vectores FM.

4.4. Resultados experimentales

La forma de evaluar el desempeño de la corrección obtenida es observando el parámetro Effective Number of Bits (ENOB). Por lo tanto, ese será el criterio mostrado en los resultados que vienen a continuación. Además, el éxito de la corrección es muy fácil de mostrar ya que el valor obtenido de ENOB se tiene que acercar al valor teórico.

La efectividad de observar solamente el parámetro ENOB se basa en que éste involucra todos los errores posibles que puedan ocurrir dentro del dispositivo. El ENOB, según la literatura, se define de la siguiente manera:

$$ENOB = \frac{SINAD - 1{,}72}{6{,}02} \tag{4.7}$$

A continuación se desarrollan los ejemplos prácticos realizados con los datos obtenidos en el laboratorio. Su procesamiento se realizó en Matlab utilizando los datos del laboratorio logrados con el dispositivo ADC comercial.

Señales sin filtrar

A modo de ejemplo, se aplicó el método de corrección donde las señales utilizadas no fueron filtradas. Es decir, no se ve afectada la salida por la transferencia del dispositivo ADC. La señal utilizada fue el vector FM con 32 tonos. La corrección se realizó para las dos bases (mencionadas en 2.1). La estructura del PWL se seleccionó de manera diferente en ambos casos.

La forma de ejecutar el experimento consiste en realizar la identificación y luego, en la etapa de validación, se evalúa cada tono por separado eligiendo datos distintos a los utilizados en el vector FM. Así se observa cómo la estructura PWL puede mejorar el ENOB resultante.

La Figura 4.16 muestra el caso para una estructura PWL de 4 dimensiones con 4 divisiones por dimensión. La identificación fue realizada en la base estándar por mínimos cuadrados.

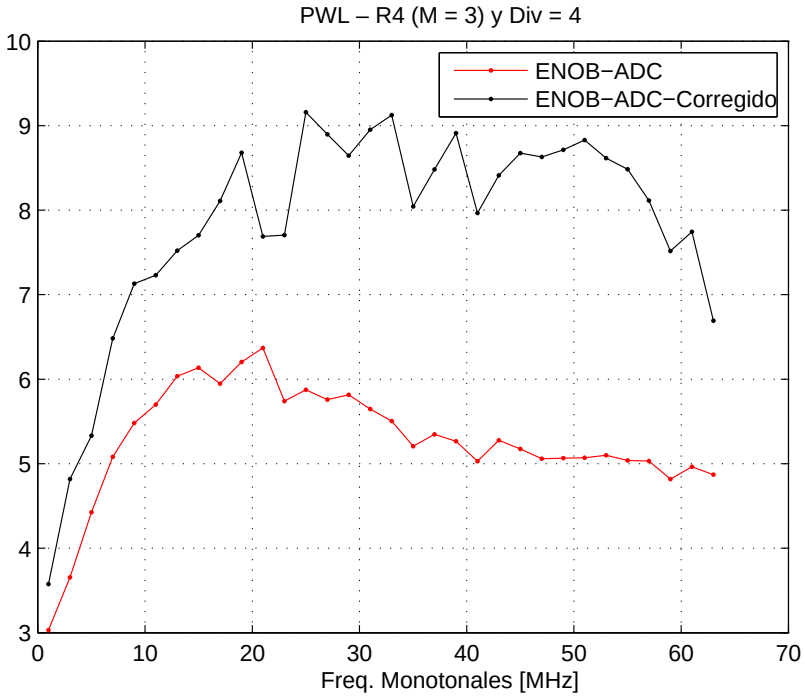

Figura 4.16: ENOB del ADC corregido. Se entrenó con un FM de 32 tonos para la base estándar. La validación consiste en correr datos distintos para los mismos tonos. Se ve que el SINAD es menor al requerido para identificar el error de la transferencia del ADC, por lo tanto la corrección se produce para el generador. La estructura PWL era de 4 dimensiones por 4 divisiones cada una. Fue realizado en la base estándar.

A continuación se muestra un ejemplo de corrección usando un regresor FIR en la base general. El resultado de la validación se ilustra en la Fig. 4.17. .

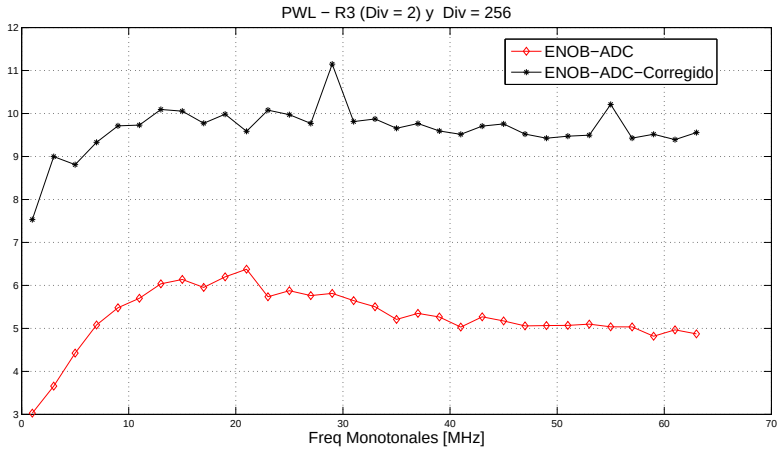

Figura 4.17: ENOB de la respuesta en frecuencia obtenida para los 32 tonos individuales. El vector de entrenamiento fue el FM. Se puede observar la característica del generador. La estructura PWL es un R3 con 256 divisiones por cada variable.

En ambos ejemplos, se puede ver cómo el método arroja valores de ENOB que llegan al valor teórico. Además, se observa que la corrección se aplica al generador ya que debido al nivel de SINAD, el ADC es transparente. También, se puede ver que mejorando la precisión de la estructura PWL se obtiene una mejor corrección.

Otra manera de visualizar la corrección obtenida es observando el espectro en frecuencias de la señal de salida con corrección y sin corrección. Los espectros para una monotonal son mostrados en la Fig. 4.18. Finalmente, los armónicos son eliminados cuando se aplica la corrección.

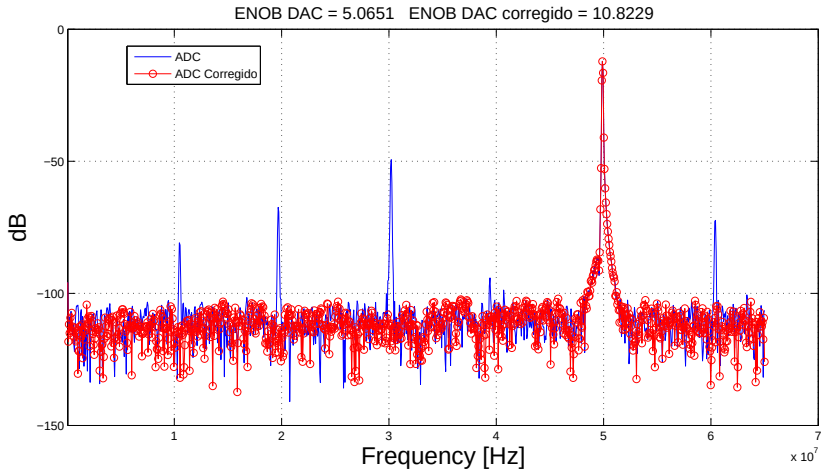

Figura 4.18: Se observa como la corrección elimina los armónicos espurios que se ven a la salida.

Señales filtradas

A los efectos de asegurar un SINAD que ponga en evidencia la transferencia del dispositivo ADC, se procedió a filtrar las señales de entrada.

Una manera de encarar este tipo de ejemplo, ante el desconocimiento de la transferencia del dispositivo, es hacer una modificación de la estructura PWL en cuanto a sus dimensiones y divisiones en el proceso de identificación, para luego verificar cual es el resultado obtenido en la etapa de validación.

Un primer ejemplo se realizó con una monotonal filtrada cuyo resultado de la corrección se muestra en la Fig. 4.19. La identificación fue realizada en la base estándar utilizando el método de mínimos cuadrados. Es importante mencionar que la cantidad de parámetros involucrados en la identificación crece de manera exponencial cuando se realizan variaciones de la estructura PWL.

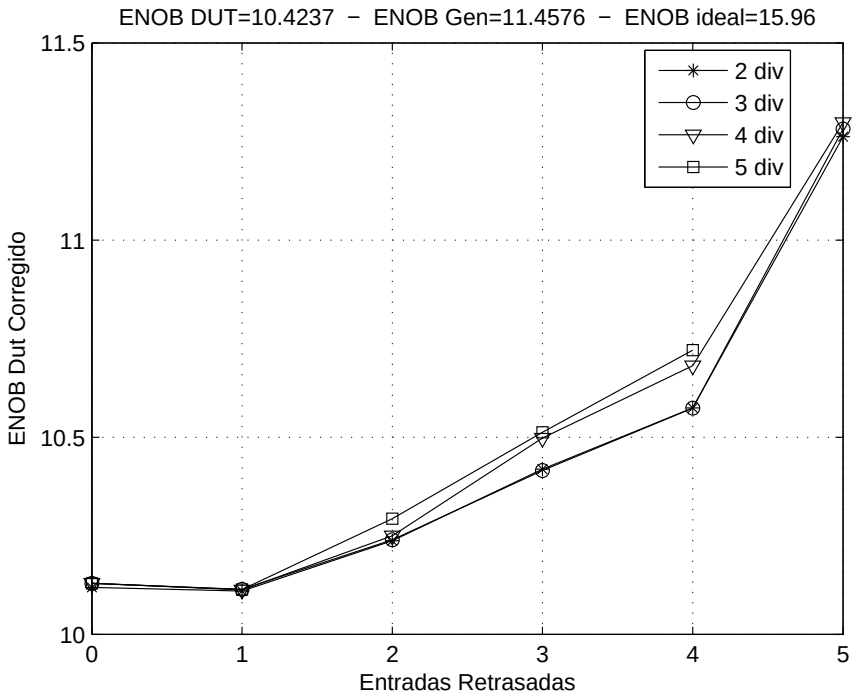

Figura 4.19: ENOB obtenido para una monotonal filtrada. Vemos el cambio en la pendiente, esto se debe que cuanto mayor el orden del modelo PWL en cuanto a dimensiones y divisiones, más datos hacen falta para un buen entrenamiento. Es decir, se perjudica la generalización. Las datos de entrenamiento son distintos a los datos de validación

Luego se utilizó la base general para el proceso de identificación. Para poder resaltar las características de la pobre generalización de esta base localizada, se excluyó un tono de la parte de entrenamiento, el elegido fue el tono correspondiente a 21 MHz, y se procedió a realizar la validación con dos grupos de datos: a) para datos coincidentes entre identificación y validación,

mostrado en la Fig 4.20, y luego b) para datos distintos entre identificación y validación, esto último se ilustra en la Fig. 4.21.

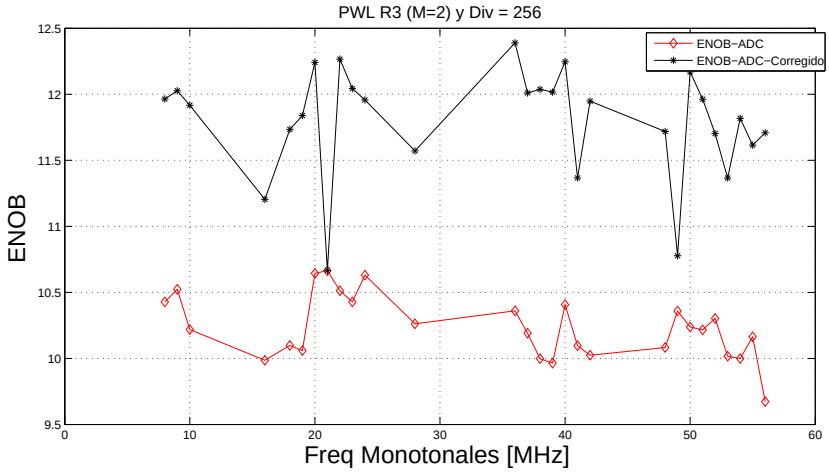

Figura 4.20: ENOB de la respuesta en frecuencia para tonos filtrados. Los datos de entrenamiento y validación son los mismos y es para poder evaluar la eficiencia del entrenamiento. El tono de 21 MHz no se incluyó en el entrenamiento y se observa que no fue corregido por el algoritmo.

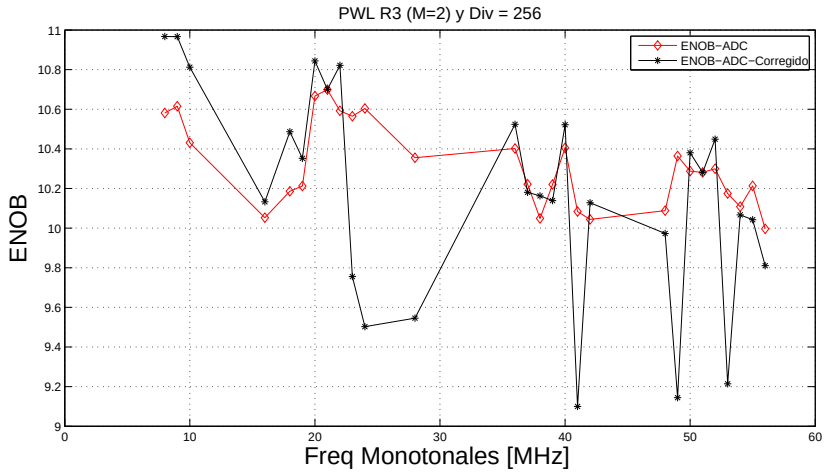

Figura 4.21: ENOB de la respuesta en frecuencia para tonos filtrados. Los punto elegidos para validación son distintos a los puntos de entrenamiento. Se puede ver la pobre generalización de esta base.

Observando las figuras 4.20 y 4.21 se destaca que la base generalizada presenta una generalización pobre. Esto se refleja en dos hechos: que el tono de 21 MHz no se corrigió en absoluto y cuando los puntos de identificación y validación son diferentes, el método empeora la salida del dispositivo ADC. La ventaja de esta base es que el método de identificación no insume recursos de memoria para su obtención. Pero, por otro lado, se requieren más señales que abarquen todos los modos, como se indica en la sección 4.3, para asegurar de mejor manera posible la general-

ización. De todas maneras, se pudo comprobar que se puede mejorar la calidad de salida de un dispositivo ADC utilizando una estructura PWL como método de post-compensación.

4.5. Definición de una arquitectura

Los ejemplos anteriores se realizaron en una plataforma de PC, donde la precisión y digitalización corresponde a la utilizada por el sistema operativo, que es de 64 bits de punto flotante. Luego se hizo una estimación de cual sería la arquitectura de un chip que implemente la estructura PWL. Esta definición se refiere a establecer la cantidad de dimensiones, la cantidad de divisiones, la cuantificación de los parámetros y cantidad de bits de la parte fraccionaria. La complejidad determinaría la cantidad de memoria física a utilizar y, por ende, marcaría un limite de la velocidad final con la cual se puede ejecutar la corrección.

Para el caso en el cual no se conoce la función de transferencia, la cantidad de dimensiones y divisiones se obtuvo por prueba y error. Los parámetros de diseño dados por la cantidad de dimensiones y divisiones se pudieron obtener conjuntamente debido a la facilidad de variar estos parámetros en las rutinas en el Matlab .

Número de Dimensiones

La manera de estimar la cantidad de dimensiones es a través de la observación de la corrección lograda cuando se aumenta la cantidad de muestras hacia atrás de la señal. Para esto se utilizó una señal monotonal y la identificación de los parámetros de la estructura PWL se obtuvieron por mínimos cuadrados en la base estándar de manera de evitar una pobre generalización. La Fig. 4.22 muestra el valor del ENOB obtenido para distintos valores de dimensiones y divisiones.

Número de Divisiones

Para este parámetro de diseño se utilizó la base general y se definió un nuevo experimento que consistió en ver como el modelo FIR implementado en la estructura PWL ajustaba el error existente ente los dispositivos ADC real e ideal. En otras palabras, se observó el éxito del modelo FIR en reproducir el error entras ambas funciones transferencias. De esta nuevo experimento surge un error definido entre lo obtenido por el modelo FIR, luego de la identificación, y error existente entre los ADC.

La manera práctica de medir este error es tomando el valor ECM de todos los puntos de entrenamiento luego de una cierta cantidad de iteraciones. [1] A continuación se muestran cómo dan los valores ECM para variaciones de: dimensiones y divisiones [2] en las Figuras: 4.23, 4.24 y

[1] El valor de las iteraciones es conocido como Epoch en la bibliografia especifica.

[2] Por limitación de memoria de la PC no se puede simular gran número de divisiones y dimensiones en el misma ejecución.

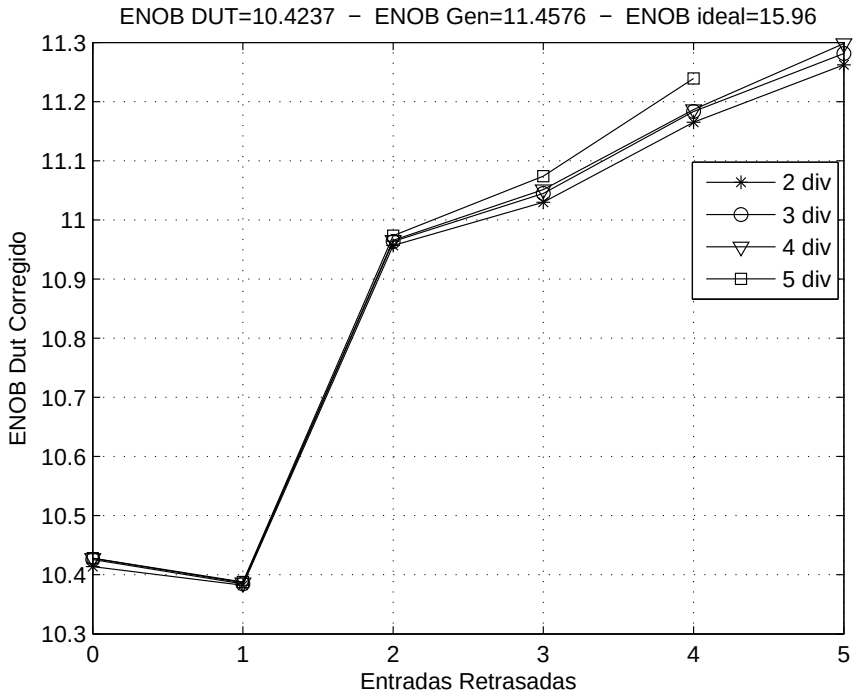

Figura 4.22: Valores de ENOB obtenidos en la corrección de un ADC para distintos valores de dimensiones y divisiones del PWL. Utilizando datos de validación distintos a los datos de entrenamiento.

4.25. La cantidad de iteraciones y el número de datos se mantuvo constante.

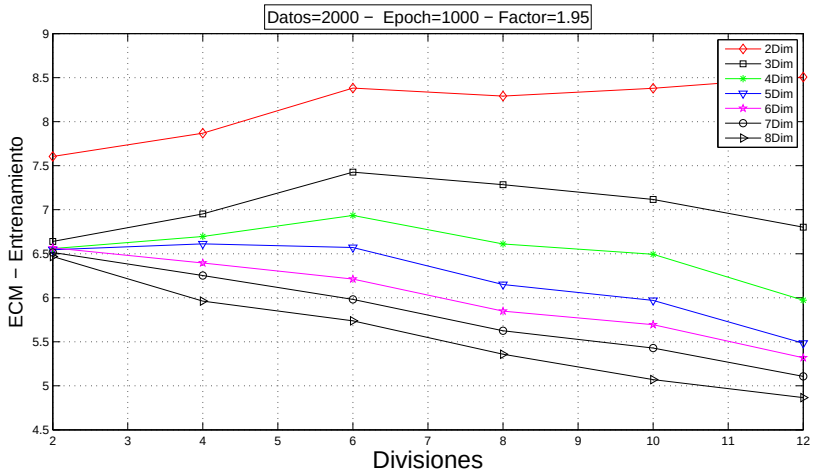

Figura 4.23: Error calculado en función de la cantidad de dimensiones y divisiones. Es el error que comete el modelo PWL en ajustar la diferencia entre el ADC real y el ideal. La cantidad de dimensiones llega a 8 mientras que las divisiones se acotó a 12.

Es claro que cuando el error estimado por la estructura PWL más se asemeje al error verdadero entre los dispositivos ADC, mejor será la corrección.

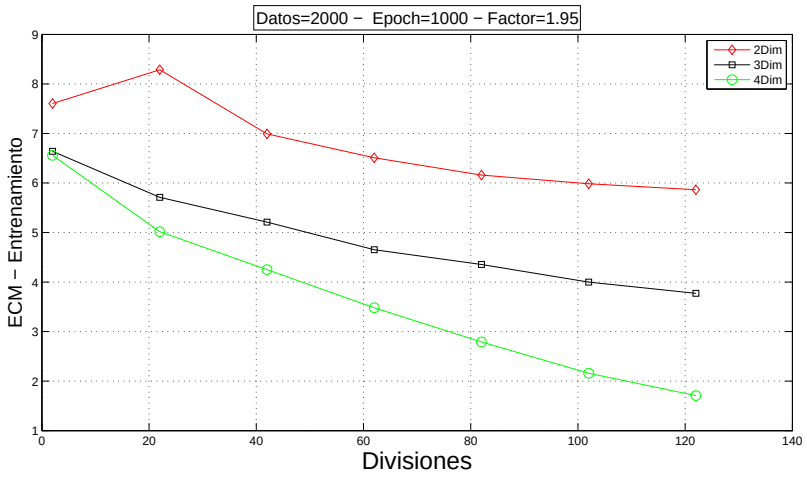

Figura 4.24: Error calculado en función de la cantidad de dimensiones y divisiones. Es el error que comete el modelo PWL en ajustar la diferencia entre el ADC real y el ideal. Se aumenta el número de divisiones y se reduce la cantidad de dimensiones.

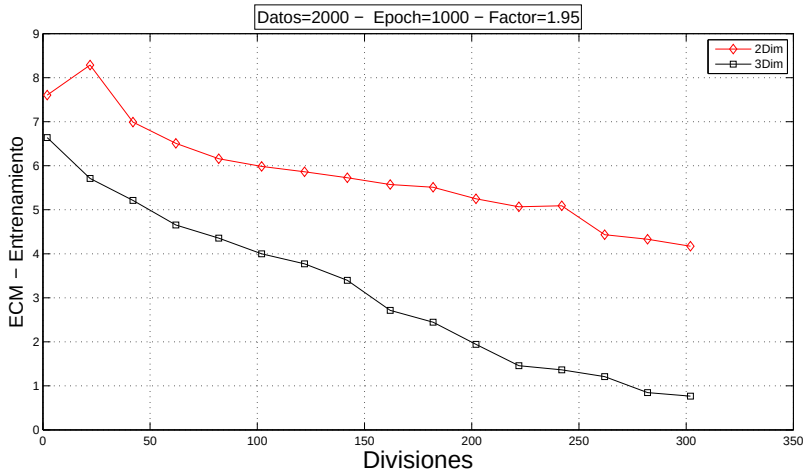

Figura 4.25: Error calculado en función de la cantidad de dimensiones y divisiones. Es el error que comete el modelo PWL en ajustar la diferencia entre el ADC real y el ideal. Se aumenta en número de divisiones y se redujo en número de dimensiones.

Si bien la forma correcta de estimar la cantidad de parámetros de retardos del FIR y la cantidad de divisiones sería midiendo el poder de corrección en validación, este procedimiento se hace complejo por las siguientes cuestiones:

- Tamaño de la matriz en la base estándar: Es cierto que la generalización es buena, pero el tamaño de las matrices multidimensionales hacen que el programa Matlab se corte por limitación de la memoria. Además, cabe recordar que los datos de entradas son obtenidos con monotonales filtradas por lo tanto el vector final FM es de un tamaño considerable.

- Generalización pobre de la base general: En este punto el tamaño de los vectores influye menos que en la base estándar pero al ser una base localizada no se aportan datos donde no existe recorrido de los datos de entrada. Por lo tanto, si uno se sale de los datos de entrenamiento la corrección empeora el resultado del ADC.

Cuantización de los parámetros del modelo

Esta cuantización se refiere al número de bits que se usa para digitalizar los parámetros del modelo. Se trata de evaluar que el error introducido por esta cuantización no perjudique la corrección final del ADC. [3] Para este parámetro de diseño, se decidió un nuevo experimento que permitiera indicar directamente la influencia de esta cuantización en el resultado final. Para esto se consideró una identificación con un número fijo de dimensiones y divisiones. Además, se utilizó la precisión de la PC para la parte fraccionaria.

Luego, se varió la cuantización de los parámetros para poder apreciar el error producido. Para poder evaluar el impacto de la cuantización, directamente, se utilizó la misma secuencia de datos utilizada en el entrenamiento en la etapa de validación. La Fig. 4.26 muestra como es el ENOB para diferentes cuantizaciones de los parámetros. En otras palabras, a medida que se cuantiza los parámetros del modelo, se empeora el desempeño de la corrección.

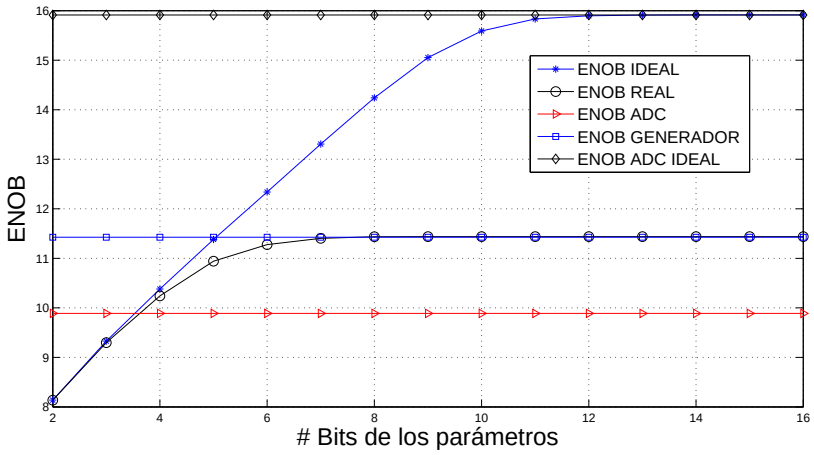

Figura 4.26: ENOB vs el número de bits que se usa para cuantizar los parámetros del modelo almacenados en la memoria. Se muestran los dos máximos teóricos correspondientes al ENOB del generador ideal y al generador considerando sus armónicos. A medida que se agregan bits a la cuantización se mejora la corrección.

Se observa que con 8 bits la validación coincide con el máximo teórico, para el caso del generador que incluye los armónicos. Para el caso del generador ideal, se considera 16 bits que, comúnmente, presentan los dispositivos de memoria comerciales.

[3] Estos parámetros son almacenados en memoria, por lo tanto no existen muchas opciones cuando se trata de memorias comerciales.

Número de bits de la parte fraccionaria

En esta sección se evalúa la cantidad de bits a utilizar en la parte fraccionaria de las variables de entrada. Como se explica en [56], este parámetro se encuentra relacionado con la cantidad de bits que se usa para los parámetros. Por lo tanto, dada la cantidad de bits de los parámetros, se agregan los bits para la parte fraccionaria. De esta forma la parte fraccionaria queda establecida en 10 bits y 18 bits según corresponda.

De todo el análisis realizado anteriormente, se puede ver que una arquitectura capaz de poder corregir un ADC implica una complejidad elevada en cuanto a: la cantidad de bits de cuantización de los parámetros, las dimensiones y la cantidad de divisiones. Se debe notar que la corrección de un ADC es en tiempo real y que demanda velocidades del orden de los cientos de Mega Hertz(MHz) con lo cual el modelo PWL y el dispositivo de memoria tienen que estar optimizados de manera de cumplir con estos requerimientos de velocidad. Por otro lado, cuanto mayor sea el dispositivo de memoria más comprometido es su tiempo de respuesta y esto va en demérito del desempeño requerido.

4.6. Conclusiones

En base a la caracterización de un dispositivo ADC comercial con un valor de ENOB distinto al valor teórico se pudo comprobar un método para la corrección post-compensador utilizando un modelo FIR implementado con una estructura PWL.

Se demostró el compromiso de identificación para el modelo FIR en cuanto el requerimiento de las señales con respecto al valor del SINAD y su riqueza para recorrer la mayor cantidad de modos de manera de garantizar la generalización. Se presentaron las ventajas y desventajas de las bases aplicadas a los dos métodos de identificación: mínimos cuadrados y método del gradiente.

Se exhibió la funcionalidad del método de compensación caracterizando su grado de éxito basado en un parámetro muy propio de un dispositivo ADC. Esto se realizó mediante ejemplos prácticos basados en datos reales medidos en laboratorio. Además, se planteó una posible arquitectura en cuanto a la definición de las características de una estructura PWL digital.

Capítulo 5

Precisión de la Representación

5.1. Introducción

En este capítulo se presenta un análisis de los errores de cuantificación para la realización práctica de una función lineal a tramos PWL. Se desarrolla una metodología para definir el tamaño de la subdivisión del dominio con el fin de lograr un error específico máximo.

Dicho análisis tiene por objetivo determinar la configuración de un motor PWL integrado en un microchip teniendo en cuenta consideraciones de tamaño de silicio, la velocidad de procesamiento y la precisión de la función que se quiere representar.

Para el análisis se asume que cada dimensión está dividida usando **T** bits con un cierto formato de entrada mostrado en la Fig. 5.1. Donde **N** son los bits que se utilizan para dividir el eje en regiones, o parte entera y **M** es la cantidad de puntos que se consideran en cada región o parte fraccionaria. **Q** es la cantidad de bits utilizada para representar los valores de los parámetros almacenados en memoria y estos parámetros serán los valores que toma la función, que se quiere aproximar, en los vértices.

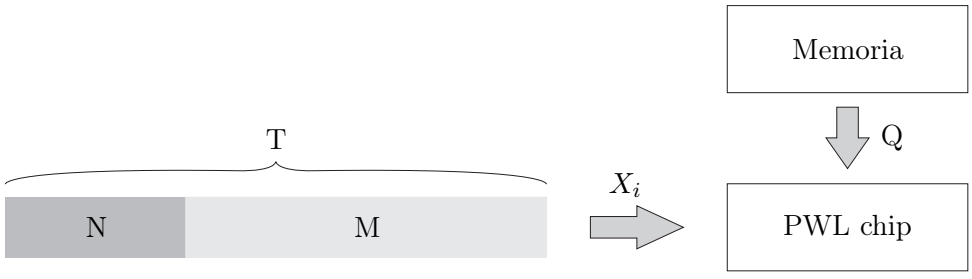

Figura 5.1: Digitalización de las entradas y salidas: N bits define las 2^N subdivisiones de cada eje en simplices; M es el número de bits de la parte fraccionaria de cada variable de entrada; Q es la cantidad de bits de que se usa para cuantizar la función evaluada en los vértices, que son los parámetros de la estructura PWL. x_i son las variables de entrada al modelo PWL.

Durante el desarrollo de este capítulo, se hace una breve mención de los dispositivos de

memoria en cuanto a los tamaños y tipos ya que en estos se almacenan los parámetros del modelo PWL. Seguidamente, se muestra un método práctico para establecer la cantidad de divisiones mínima que garantiza una cota de error máxima entre la función no lineal y la aproximación que se obtiene mediante el uso de la función PWL. Luego se presenta un análisis de los dos errores que se comenten debido a la cuantización: de los vértices o parámetros y de la parte fraccionaria. A continuación, se da una relación de compromiso entre ambos para que el aporte sea equitativo entre ambas representaciones digitales. Por último, se enuncian la conclusiones referentes al capítulo.

Breve relación de los tamaños de memoria

En una aplicación PWL el tamaño (L) de la memoria crece en forma exponencial según la siguiente formula:

$$L = 2^{(N * n)} \tag{5.1}$$

donde n corresponde a la cantidad de variables del sistema. A modo de ejemplo, si $n = 5$ y $N = 6$ el tamaño de la memoria sería $1,073$ GB, suponiendo que los parámetros sean de un Byte.

Generalmente el número de dimensiones esta relacionado con la ecuación física que se quiere aproximar usando la estructura PWL. Por otro lado, el tamaño de los dispositivos de memoria son los disponibles en el mercado y estos no se pueden cambiar a menos que se haga un diseño personalizado. Además, estos dispositivos ya vienen con ciertas configuraciones establecidas entre tamaño y ancho de palabra. Por ejemplo: 256 Mbits presenta una configuración de 32 M de direcciones de 8 bits de ancho de palabra.

Otro factor importante en el tamaño de la memoria es la posibilidad de su integración en el chip sin tener que recurrir a dispositivos externos, por ejemplo: Double Data Rate (DDR), Synchronous Dynamic Random-Access Memory (SDRAM), ya que el retardo (*latency*) implica complicaciones en el diseño si se quieren alcanzar velocidades de procesamiento superior a los 100 KHz o si la arquitectura presenta realimentación de las salidas en las entradas ya que esto compromete la velocidad final que tendrá el dispositivo.

Como se puede apreciar, solamente el parámetro N es el que se puede fijar para un diseño dado. Por lo tanto, es el parámetro cuyo valor final tendrá un impacto importante en el diseño ya que el número de dimensiones y los dispositivos de memoria son fijados externamente.

5.2. Aproximación de una Función no lineal por su representación PWL

En esta sección se busca minimizar el número de divisiones (2^N) de manera de cumplir con la aproximación de la función no lineal. Mantener un número de divisiones pequeño, influye en el tamaño de la memoria, como se comentó previamente.

Supongamos que $f(x)$ es una función continua no lineal con $x \varepsilon R^n$. Esta función se puede escribir utilizando la serie de Taylor en un entorno de un punto arbitrario x_0, de la siguiente manera:

$$\begin{aligned} f(x) =& f(x_0) + \frac{1}{1!} \bigtriangledown f(x_0)(x - x_0) + \\ &+ \frac{1}{2!}(x - x_0) H f(x_0)(x - x_0) + \dots \end{aligned} \tag{5.2}$$

donde $\bigtriangledown$ es el operador gradiente y H es la matriz Hessiana. Si se realiza una aproximación de primer orden, en un entorno de x_0, tomando hasta el primer término de la expansión de Taylor, la ecuación se reduce a los siguiente:

$$f_{pwl}(x) = f(x_0) + \frac{1}{1!} \bigtriangledown f(x_0)(x - x_0) \tag{5.3}$$

La Figura 5.2 muestra la ilustración gráfica de $f(x)$ y $f_{pwl}(x)$ de la Ec. 5.3 y gráfica el error resultante de la aproximación. Utilizando las Ec.(5.2) y Ec.(5.3), el error se puede representar

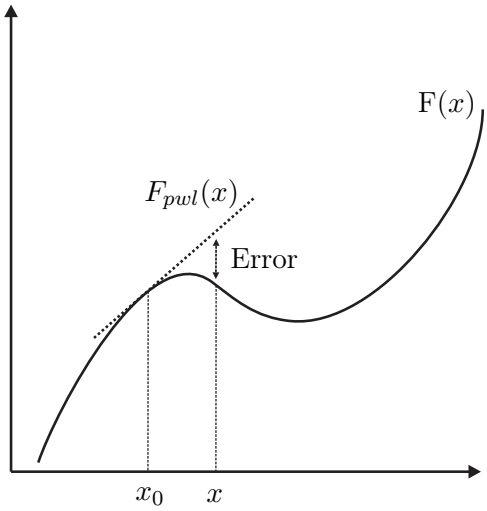

Figura 5.2: Aproximación Taylor de $f_{pwl}(x)$ en un entorno de x_0 y representación del error entre $f_{pwl}(x)$ y f(x).

de la siguiente manera:

$$\begin{aligned} \xi &= f(x) - f_{pwl}(x) \\ \xi &= H f(x_0)(x - x_0)^2 + \dots \end{aligned} \tag{5.4}$$

Si consideramos que $x - x_0$ es lo suficientemente pequeño, podemos despreciar los términos de

orden mayor en la expansión de (5.4). Bajo esta suposición, el error es representado por:

$$\xi = (x - x_0)^T Hf(x_0)(x - x_0) \tag{5.5}$$

donde H tiene la siguiente formula:

$$H(f) = \begin{bmatrix} \frac{\partial f^2}{\partial x_1 \partial x_1} & \frac{\partial f^2}{\partial x_1 \partial x_2} & \cdots & \frac{\partial f^2}{\partial x_1 \partial x_n} \\ \frac{\partial f^2}{\partial x_2 \partial x_1} & \frac{\partial f^2}{\partial x_2 \partial x_2} & \cdots & \frac{\partial f^2}{\partial x_2 \partial x_n} \\ \vdots & & & \vdots \\ \frac{\partial f^2}{\partial x_n \partial x_1} & \frac{\partial f^2}{\partial x_n \partial x_2} & \cdots & \frac{\partial f^2}{\partial x_n \partial x_n} \end{bmatrix} \tag{5.6}$$

Y la norma del Hessiano esta definida según la Ec. 5.7:

$$\|H(x_0)\| = \sqrt[2]{(\frac{\partial f^2}{\partial x_1 \partial x_1})^2 + (\frac{\partial f^2}{\partial x_1 \partial x_2})^2 + \cdots + (\frac{\partial f^2}{\partial x_n \partial x_n})^2} \tag{5.7}$$

Por lo tanto, el error de aproximación queda limitado de la siguiente manera:

$$\begin{aligned} |\xi| &= \left\| \frac{1}{2!}(x - x_0)^T Hf(x_0)(x - x_0) \right\| \\ |\xi| &< \left| \frac{1}{2} \right| \left\| (x - x_0)^T \right\| \|Hf(x_0)\| \|(x - x_0)\| \end{aligned} \tag{5.8}$$

Si se quiere limitar el error cometido por una constante α, se impone la restricción de la siguiente manera:

$$|\xi| < \left| \frac{1}{2} \right| \|(x - x_0)\|^2 \|Hf(x_0)\| \leq \alpha \tag{5.9}$$

Utilizando la ecuación anterior podemos obtener el tamaño del entorno alrededor de x_0 donde el error va a estar acotado dentro de los limites impuestos por la cota α

$$\rho < \sqrt{\frac{2\alpha}{max_{x_0} \|Hf(x_0)\|}} \tag{5.10}$$

Donde $\rho = \|(x - x_0)\|$.

Por lo tanto, la Ec. (5.10) muestra que el valor máximo de la norma del Hessiano nos permite evaluar el tamaño de la grilla para el peor caso. O sea que x_0 se asume como el punto en el espacio del dominio donde la norma de la matriz Hessiana es máxima. Una vez detallado el método teórico, el objetivo es encontrar de manera práctica el valor máximo del Hessiano. Para

esto se utilizan técnicas de optimización [1].

A las rutinas de optimización se les proporciona un valor inicial cualquiera, luego el algoritmo evoluciona dando como resultado el máximo valor. Si bien el método de optimización es garantizado por el paquete propio del Matlab, no se garantiza que el máximo logrado sea un máximo global.

A continuación se describe la forma de obtener un máximo global utilizando el mismo paquete básico de optimización. El procedimiento consistió en dividir el dominio de la función con una grilla uniforme arbitraria. Luego se evaluó la norma del Hessiano en todos los puntos de la grilla y se obtuvo el valor máximo. Una vez logrado ese máximo, se tomaron estas coordenadas como punto inicial para el paquete de optimización. En otras palabras, la grilla da una idea de donde puede estar el máximo y con la optimización se obtiene el máximo global. Este método del grillado arbitrario se repite, por lo menos dos veces, aumentando la cantidad de divisiones por dos. Cabe destacar que la manera de evitar máximos locales es un método práctico, con lo cual se puede iterar hasta poder asegurar el máximo del Hessiano.

Una vez obtenido el máximo del Hessiano, el próximo paso es el cálculo del número de divisiones de la grilla, este es dado por:

$$Div = \frac{Range(X)}{2\rho} \tag{5.11}$$

El factor 2 que aparece en el denominador proviene del hecho de que el entorno donde el error está acotado tiene un radio ρ (y un diámetro 2ρ). El rango será el máximo y mínimo de cada intervalo.

Una vez que el número de divisiones es calculado se tiene que plantear el algoritmo para encontrar los parámetros del modelo PWL. La función PWL va a ser, en general, distinta a la linealización de primer orden de $f(x)$ en x_0. Por lo tanto, el tamaño de la grilla de la Ec. (5.11) tiene que ser interpretado como una cota mínima del número de divisiones. En otras palabras, la implementación en silicio tiene que tener al menos el número de divisiones calculado anteriormente.

Si bien el método da el número de divisiones para cada dimensión, la implementación de una estructura PWL es más sencilla si las divisiones son iguales para todas las dimensiones o variables de entrada. Por lo tanto para obtener una grilla uniforme se aplicaron transformaciones lineales para mapear las variables de entrada a un hipercubo unitario $[0\ 1]^n$.

Finalizado el paso anterior, el número de divisiones se acomodó a una implementación digital. Por lo tanto, el número de divisiones se cuantificó en bits de manera de aplicarse sencillamente a un diseño digital. El número de divisiones será de la forma 2^N, donde N ahora es el número de bits, mencionado anteriormente.

[1] Las rutinas de optimización se encuentran en el paquete de optimización del Matlab. Es decir, que el valor máximo se determinará de manera numérica.

Ejemplificación del método.

A continuación se presentan varios ejemplos prácticos para mostrar el funcionamiento del método propuesto. En cada ejemplo se muestra la ecuación no lineal, el error máximo exigido, su aproximación PWL (cuando se puede realizar el gráfico) y el error arrojado en la fase de comprobación. Esta fase consiste en elegir puntos de manera aleatoria para ser evaluados en la función no lineal y su representación PWL. Es claro que todos los puntos tienen que estar por debajo del error exigido.

- Ejemplo en R^1: El primer ejemplo consiste en la siguiente función:

$$f(x) = \sec(x) * 2^x + \exp(-\frac{x}{8}) \tag{5.12}$$

  El rango de la variable a considerar es $[-8\ \ 8]$. La Fig. 5.3 muestra la gráfica de la función. La Figura 5.4 muestra la representación PWL y el error obtenido. La comprobación utilizó 10K puntos aleatorios distribuidos en el rango [0 1].

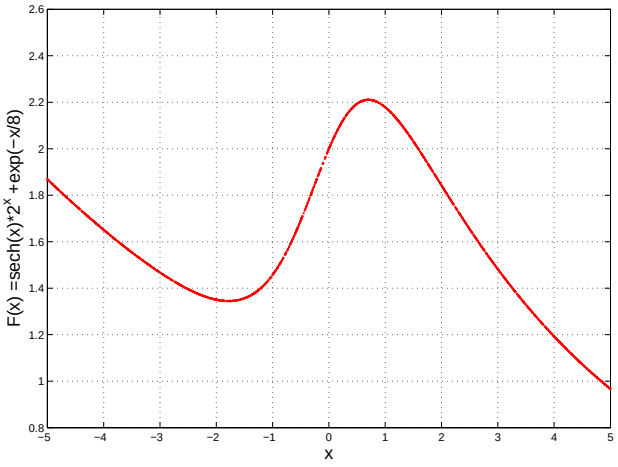

Figura 5.3: Función no-lineal en R^1. Se muestra como es la no linealidad de la función que se tomo como ejemplo.

- Ejemplo en R^2: En este caso se utiliza la siguiente función de R^2:

$$f(x, y) = \sin(x^2) * (-x^y) \tag{5.13}$$

  La Figura 5.5 muestra la gráfica de la función y la comprobación se muestra en la Fig. 5.6. En este caso se gráfica utilizando 2K puntos aleatorios para mantener el tamaño del gráfico acotado, pero para la comprobación efectiva se utilizó 50 K puntos. El número de divisiones arrojado por el método fue 202, pero llevado a una implementación digital, el número final utilizado fue $2^8 = 256$.

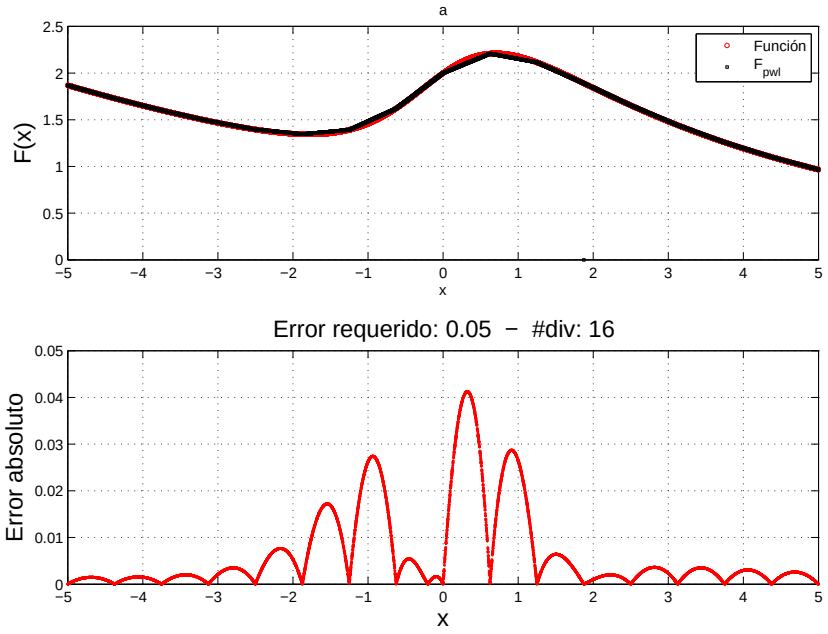

Figura 5.4: a) Gráfica de la función y su aproximación PWL usando 16 divisiones para el dominio. b) Resultado de la comprobación, donde se ve que el error está por debajo del error exigido.

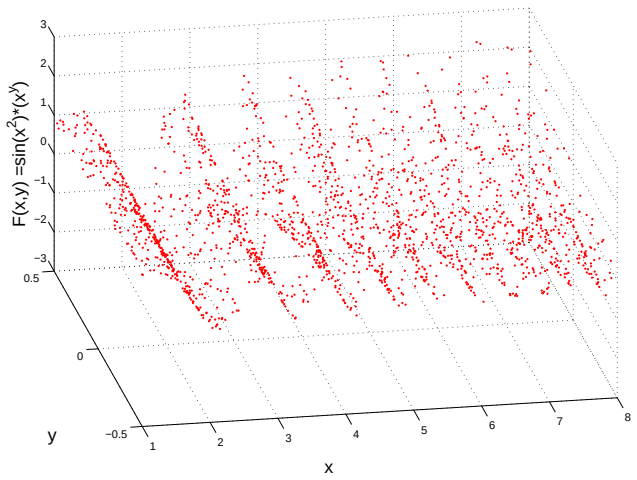

Figura 5.5: Evaluación de una función no lineal de R^2. Se evalúa una cierta cantidad de puntos para dar una idea de la no linealidad de la función. Más adelante se ver como esta función es representada por una función PWL

- Ejemplo en R^4: Para este caso se eligió la siguiendo función:

$$f(x, y, z, w) = \frac{y^2 * \sin(x)}{z} * \cos(w) \tag{5.14}$$

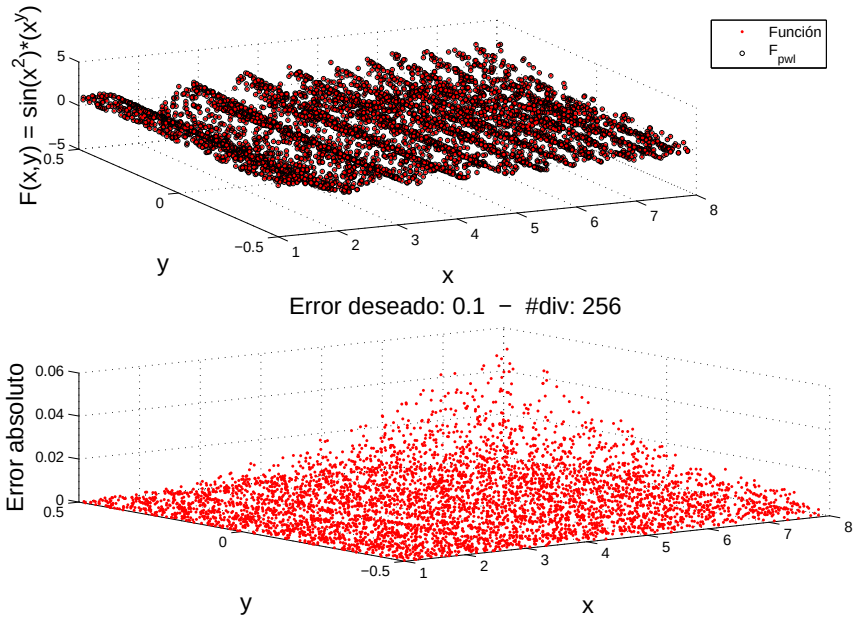

Figura 5.6: a) Función no lineal y su representación PWL con 256 divisiones. b) Resultado de la comprobación para 2K puntos.

La gráfica de esta función no se puede realizar por ser de R^4, por lo tanto se muestra el resultado de la comprobación en la Fig. 5.7 para 2 K puntos aleatorios. Nuevamente, se utilizó 50K puntos pero no es mostrado por una cuestión de tamaño de los gráficos.

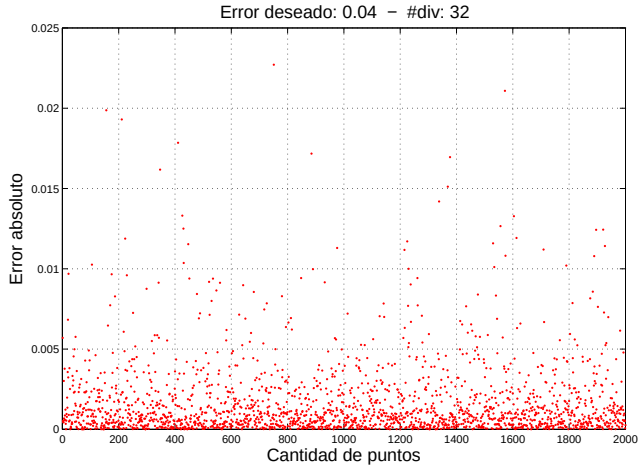

Figura 5.7: Error entre $f(x)$ y f_{PWL} para una función de R^4 no-lineal. La cantidad de divisiones es 32.

El método puede ser aplicado a sistema de mayor dimensión a expensas de un mayor tiempo de ejecución.

Comprobación práctica

Si bien las ecuaciones anteriores demuestran la utilización del método, se elaboró en Matlab una rutina, de manera de poder verificar los resultado para un conjunto de funciones no-lineales cualesquiera.

Para esto, se elaboró un lista de 49 funciones elegidas arbitrariamente pertenecientes a R^2 y R^4. Para estas funciones se automatizó el método y se corrieron simulaciones para ver el error obtenido entre la función y su aproximación PWL con el número de divisiones encontrado por el método. El cálculo del error se hizo con un conjunto de 50K puntos aleatorios y se verificó que el error sea menor al establecido. Este conjunto de funciones arbitrarias se ejecutó para distintas cotas de error, arrojando un resultado exitoso ya que siempre el error entre la función y su aproximación se mantuvo acotado por este error establecido. Además, se comprobó que disminuyendo la cantidad de divisiones que arrojó el método, el error superaba la cota establecida.

A modo de ejemplo se listan algunas de las funciones en R^2 utilizadas.

$$\begin{aligned} f_1(x,y) &= log(y+2)\,10\,sin(x) \\ f_2(x,y) &= 100\,(y-x^2)^2 + (1-x)^2 \\ f_3(x,y) &= \arctan(20\,x)\,sin(y) \\ f_4(x,y) &= sin(x\,y)^2\,y^4 + 2\,(-sin(x\,y)\,x\,y + cos(x\,y))^2 + sin(x\,y)^2\,x^4 \end{aligned} \tag{5.15}$$

5.3. Error de los parámetros representados en memoria

En la sección anterior se evaluó el número de divisiones por eje de coordenadas. A continuación, se explica el error que se produce por la cuantización al digitalizar los parámetros que se almacenan de la memoria.
La cantidad de bits disponibles para estos parámetros los denominamos Q. El error producido por la cuantización de estos parámetros es similar al error de un conversor analógico digital:

$$e_c = 2^{-(Q+1)} \tag{5.16}$$

La Fig. 5.8 muestra el error producido simulado en Matlab para una función de R^2. En este ejemplo se uso una función lineal de manera de que el error producido sea solamente por la cuantización. Este error se calculó de la siguiente manera:

$$e(x_i) = abs(f(x_i) - f_{pwl}(x_i)) \tag{5.17}$$

El error e(x_i) se obtuvo usando el valor medio de 1K puntos aleatorios para cada cada caso de Q. El paquete de herramientas en Matlab del PWL se manipuló de manera de garantizar la cuantización de los bits para cada parámetro de memoria.

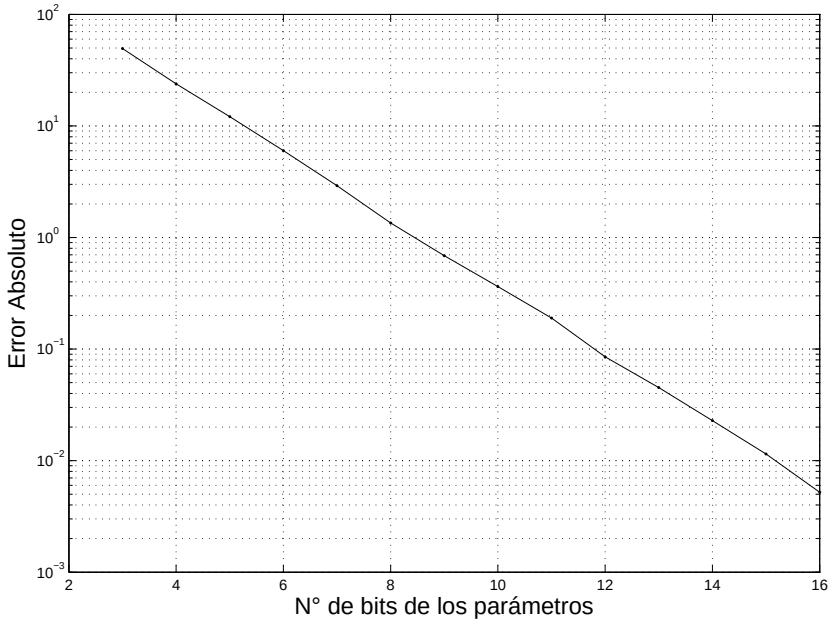

Figura 5.8: Ejemplo de una función de R^2 cuando la aproximación PWL tiene los parámetros cuantizados con distintos números de bits.

5.4. Error de la parte fraccionaria

Seguidamente, se presenta un método para determinar la cantidad de bits de la parte fraccionaria. Esto es para un diseño de propósitos generales.

Cada variable de entrada x_i tiene la siguiente representación:

$$x_i = \widetilde{x}_i * 2^{-(N+M)} \tag{5.18}$$

Y el error de cuantización está acotado por:

$$\mid \Delta x_i \mid \leq 2^{-(N+M+1)} = 2^{-(T+1)} \tag{5.19}$$

Considerando que la función de entrada fue escalada al intervalo $[0, 1]$, la derivada de la

función puede ser acotada de la siguiente manera:

$$
\begin{aligned}
| \frac{\delta f}{\delta x_i} | &\leq 2^N \\
|| \frac{\delta f}{\delta x_i} || &\leq \sqrt{n} * 2^N \\
|| \delta x || &\leq \sqrt{n} * 2^{-(T+1)} = \sqrt{n} * 2^{-(N+M+1)} \\
e_q = || \frac{\delta f}{\delta x} || * || \Delta x || &= \sqrt{n} * 2^N * \sqrt{n} * 2^{-(N+M+1)} \\
e_q &= n * 2^{-(M+1)}
\end{aligned}
\tag{5.20}
$$

La Figura 5.9 muestra el error de cuantización simulado en Matlab para el parámetro **M** para una función de R^2. La función elegida es lineal de manera de mostrar el error de la cuantización de M. Como el caso de Q, el error fue calculado utilizando la media de 1K puntos aleatorios para cada caso de M. El error se definió igual que en el caso anterior.

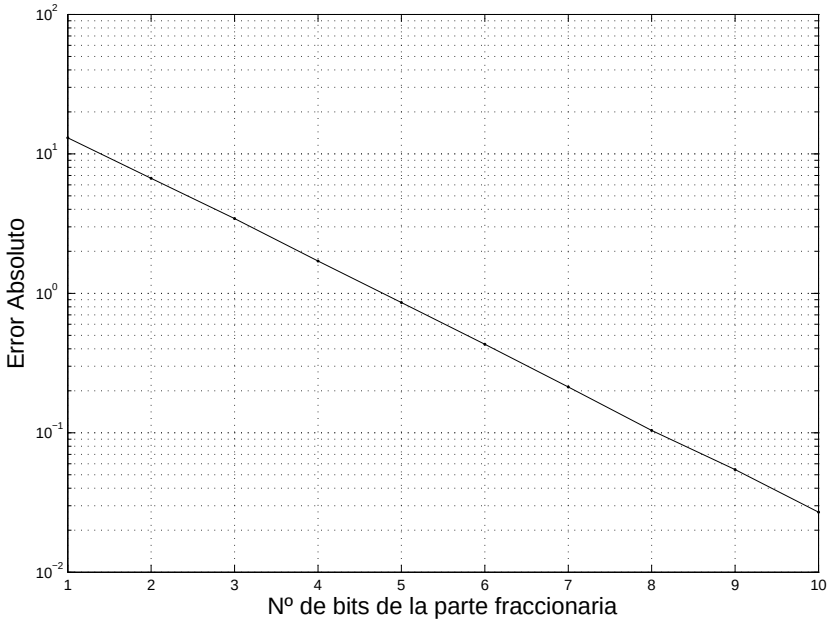

Figura 5.9: Error de la cuantización de la parte fraccionara para distintos números de bits. Simulado para una función de R^2.

5.5. Compromiso entre los parámetros y parte fraccionaria

Seguidamente, se obtiene una relación entre los números M y Q de manera de lograr que la contribución del error de ambos sea similar ya que, como se vio anteriormente, ambos contribuyen al error final. La idea es igualar ambos errores y minimizar ambos como un solo número.

Cuadro 5.1: Relaciones entre el numero de bits de los vértices (Q) y la parte fraccionaria(M) para una aplicación de propósito general.

N	M	Q	N+M
8	10	8	18
8	12	10	20
8	14	12	22
8	16	14	24
8	18	16	26

Asumamos que tenemos un tamaño fijo de memoria L y una cierta dimensión de entrada $n = 3$. Con L = $(2^N)^n$, $L = 16M$ tenemos la siguiente ecuación:

$$\begin{aligned} 2^N &= L^{\frac{1}{n}} \\ N &= \frac{1}{n} * \frac{log(L)}{log(2)} = 4/n \end{aligned} \tag{5.21}$$

Está claro que se tiene una relación de compromiso entre el tamaño de memoria L, ancho de la palabra de memoria Q, dimensión del espacio n, las divisiones del dominio (2^N) y la parte fraccionaria M. Generalmente el n esta directamente relacionado con la aplicación a resolver. Tomaremos n=3, sin perder generalidad. El N proviene de la sección anterior y el L depende del componente de memoria disponible.

Ahora, se presenta el procedimiento de evaluación del Q y M. Vale la pena mencionar que ambos parámetros se pueden elegir libremente, no obstante, el parámetro Q estará determinado por el dispositivo de memoria disponible.

Deseamos que ambas magnitudes , e_c (Eq. (5.16)) y e_q (Eq. (5.20)) tengan igual contribución al error final, por lo tanto:

$$\begin{aligned} e_c &= e_q \\ 2^{-(Q+1)} &= n * 2^{-(M+1)} \\ Q &= -\frac{log(n)}{log(2)} + M \end{aligned} \tag{5.22}$$

Esta simple relación da una idea de como elegir la arquitectura correcta para una aplicación de propósito general. Como ejemplo, si $n = 3$ y $N = 8$ (Memoria de 16MB), entonces $Q = -1{,}58 + M \approx -2 + M$. El cuadro 5.1 muestra la relación de la Ec. 5.22.

En el caso que T=24 bits, los mejores valores para M, N y Q son: N=8, M=16 y Q=14; los errores son $e_c = 0.0031\,\%$ y $e_q = 0.0023\,\%$.

La figura 5.10 muestra la simulación incluyendo la cuantización de los vértices o parámetros y de los bits que componen la parte fraccionaria. Como ejemplo, de la Fig. 5.10 se puede ver que

si tenemos 8 bits para los parámetros, el agregado de más de 5 bits para la parte fraccionaria, no reduce el error.

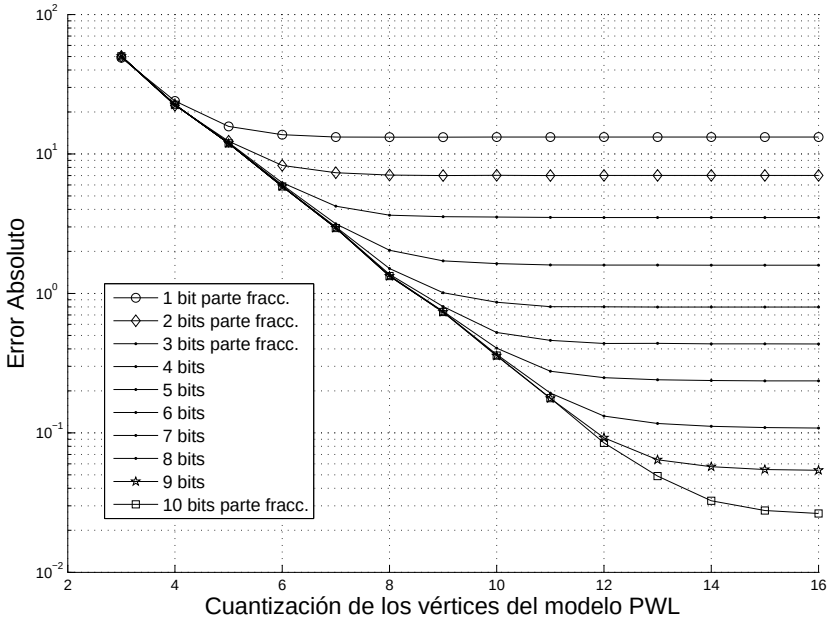

Figura 5.10: Error final considerando ambos efectos de cuantización: vértices y parte fraccionaria. Para una función de R^2. El eje x son los bits de los vértices o parámetros y la series o curbas son la cantidad de bits de la parte fraccionaria.

Para validar la simulación, se tomó una función lineal de manera que el error de aproximación PWL sea cero y se consideró el valor promedio de 1000 ensayos aleatorios para un valor dado de cuantización del parámetro y la cantidad de bits de la parte fraccionaria. En otras palabras, para cada punto de la Fig. 5.10 la rutina se ejecutó 1000 veces con valores aleatorios para una configuración fija de cuantización del parámetro y el número de bits de la parte fraccionaria.

5.6. Conclusiones

En este capítulo se abordó el tema de la memoria por su importante rol en la estructura PWL debido a su crecimiento exponencial en tamaño. Además, se expuso la relación de compromiso entre el tamaño de la memoria, las dimensiones de la estructura y la cantidad de divisiones que se elige para el dominio. Dicho compromiso surge por el hecho que la memoria es un dispositivo comercial.

Se estableció un método práctico para relajar el tamaño de memoria admitiendo un error en la aproximación dada por la estructura PWL. El método se planteó analíticamente y se comprobó de manera práctica bajo un entorno de simulación.

Se demostró, de manera simple, la forma de dimensionar la parte fraccionaria de las variables de un sistema PWL, conocido el dispositivo de memoria que se va a utilizar de manera de igualar los aportes al error de cuantización de ambos factores.

Capítulo 6

Diseño de una implementación con tasa de salida de datos optimizada

6.1. Introducción

Este capítulo presenta el diseño de una nueva arquitectura, con el objetivo de lograr mayor velocidad de procesamiento con una latencia (*latency*) mínima. En especial la reducción de la latencia influye cuando se requiere una realimentación de la salida, como es el caso de la estructura NOE.

Se consideró, como parte del diseño, que el chip pueda soportar 3 maneras de operación: con variables de entrada independientes; con variables tipo FIR, es decir, una entrada y muestras de ésta retrasadas en el tiempo y por último entradas tipo NOE. Además, se impuso que estas configuraciones se puedan programar desde el exterior. Esta imposición condujo a una clara división: un núcleo (core) PWL y una lógica de entorno que implementa los retardos y la realimentación correspondiente de la salida hacia la entrada.

Como se sabe de la literatura, el rendimiento de los sistemas digitales está estrechamente vinculado al retardo de la memoria. Los sistemas más conocidos, como los microprocesadores, utilizan una memoria interna, denominada cache, que va adquiriendo datos futuros aprovechando el orden secuencial de los programas que se ejecutan. De esta manera obtienen los datos necesarios para el cómputo rápidamente. La opción de cache quedó desestimada para el caso de la PWL ya que las direcciones generadas en el cómputo no están de forma continua en memoria, que es la suposición básica de la lectura "burst", su demostración queda expuesta en el Apéndice A, por lo tanto, se impuso que la memoria fuese integrada para asegurar el mínimo retardo. Si bien esta imposición limita las dimensiones y divisiones de la estructura PWL se hace para poder lograr las metas de optimización, mencionadas anteriormente.

El primer tema que aborda el desarrollo de presente capítulo es la noción sobre cómo se realiza el cálculo de las funciones simpliciales PWL. Seguidamente se comenta sobre los dispositivos de

memoria por su importante rol en el diseño y se explica el desarrollo de dos bloques: ADC y generador de reloj. Estos últimos no forman parte del chip, sin embargo, pero su inclusión esta vinculada a la simulación del diseño y posterior implementación y validación en FPGA.

Luego se detalla la arquitectura que se realizó de manera de poder satisfacer los requerimientos de frecuencia de operación y *latency* del sistema, y también, se contempla el cumplimiento de los diferentes modos de operación en lo que refiere al tipo de entrada posible al núcleo PWL. En esta parte, se explica el nexo entre el dispositivo de memoria y la velocidad final de procesamiento, introduciendo la noción de *pipeline* de procesamiento de los datos. A continuación se explica el diseño de cada bloque constituyente de la arquitectura. Hacia el final se presentan los resultados de las simulaciones e implementación en FPGA. Por último, se describen las conclusiones a las que se arriba.

Nociones sobre el cálculo PWL

A continuación se explica, en forma resumida, cómo se realiza la evaluación de las funciones PWL del tipo simplicial.

La representación en simplices propuesta en [57] y [41] y el método de descomposición propuesto en [9] para un dominio compacto multi-dimensional provee las claves para la implementación digital de la estructura PWL. Para un dado simplice, toda función PWL se puede expresar como la siguiente suma ponderada:

$$f_{pwl}(X) = \sum_{i=1}^{n+1} \mu_i V_i \tag{6.1}$$

donde X es un punto, perteneciente a R^n, al cual se le aplica la función PWL.

Los términos internos de la sumatoria: μ_i y V_i son los valores correspondientes a la descomposición simplicial y los parámetros de la estructura PWL respectivamente. Estos últimos representan el valor de la función evaluada en los extremos del simplice. En la literatura, estos puntos se conocen como vértices. En lo que respecta a los valores μ_i, se tienen que computar y dependen del punto de entrada (X).

En resumen, dado un punto en el espacio, su evaluación en la función PWL se obtiene realizando la suma anterior con los valores de los parámetros, luego de obtener los coeficientes μ. Los valores de los vértices se consideran existentes y se encuentran almacenados en una memoria. Las direcciones de la memoria, se obtienen a partir del punto X de entrada.

6.2. Consideraciones previas a la implementación

Antes de iniciar el proceso de diseño, se analizaron las restricciones impuestas por los siguientes bloques: los dispositivos de memoria, dispositivos de conversión analógico a digital y los generadores de reloj. El objetivo del diseño es la maximización de la velocidad de operación,

por lo tanto se establecieron las siguientes decisiones: las variables de entrada se representaron con 14 bits, donde se utilizaron 4 bits para definir las subdivisiones del dominio y los restantes 10 para la parte fraccionaria. Los 4 bits de parte entera establecen una relación de compromiso entre la precisión y el tamaño de memoria. Los 10 bits de parte fraccionaria están en relación a la representación de los parámetros en memoria, que se tomó en 8 bits.

Dispositivo de memoria

La vinculación del dispositivo de memoria a la estructura PWL es clara, según se vio en los capítulos anteriores. La velocidad y tamaño del dispositivo de memoria son variables que juegan un rol opuesto generando una relación de compromiso muy fuerte. La propuesta eS la utilización de un dispositivo de memoria integrado para demostrar el rendimiento de velocidades.

El tiempo de latencia mínimo para una memoria integrada es de un ciclo de reloj, es decir, una vez aceptada la dirección de memoria se obtiene el dato en el ciclo posterior. A modo de ejemplo, si se quieren leer cuatro direcciones de memoria, se tiene el último dato cinco ciclos de reloj más tarde, desde que se capturó la primera dirección de memoria. Esta limitación se puede subsanar replicando memoria o utilizando un dispositivo con la opción de multipuerto. De esta manera, una vez conocidos todas las direcciones de memoria se procede con la búsqueda en paralelo de dichos datos.

Funcionamiento del ADC

Dado que el chip recibe señales digitales de entrada, por lo tanto, el uso de conversores analógicos a digital es un hecho. Si bien el PWL chip tiene su propia frecuencia de operación, el diseño tiene que contemplar cuál es el elemento que tendrá como función procesar los datos analógicos para poder alimentar las entradas del chip. Para poder estimar las capacidades y especificaciones en bits de los conversores existentes se observaron dos caminos claves: dispositivos integrados y dispositivos discretos.

A continuación, el cuadro 6.1 muestra los ADC integrados disponibles en publicaciones. Estos se consideran para una posible integración del ADC dentro del chip. Se puede ver que la última opción es la relevante para la especificación.

La segunda opción fueron los ADC comerciales. Para esto se buscó en la web sobre integrados disponibles que cumplan con la característica de cantidad de bits y de frecuencia de operación. Cabe destacar que el ADC es una limitante de la frecuencia máxima de los datos que entran al PWL. Dentro de los ADC comerciales, el que se utilizó como referencia es: AD9642: 14-Bit, MSPS/250, 1.8 V de Analog Device. Una consideración de diseño fue contar con tantos dispositivos ADC como entradas tenga el chip de manera de evitar la conversión serie paralelo.

Para estudiar el sincronismo ente el chip y el dispositivo ADC se verificó la hoja de datos, mostrado en la Fig. 6.1. Esta muestra un diagrama temporal de un ADC comercial destacando

Cuadro 6.1:
ADC integrados obtenidos de las publicaciones de la IEEE.

Bits	Frec. [MS/s]	Arq.	Tecnología .	Fecha
8	250	pipeline	0,13u	oct-11
10	50	SAR	0,13u	oct-11
12	50	pipeline	65n	oct-11
14	100	pipeline	0,18u	sep-09
10	100	SAR	65n	oct-11
6	2000	–	0,18u	oct-09
3	4250	Flash	0,13u	nov-11
14	50	pipeline	0,13u	nov-11
10	500	pipeline	0,13u	jun-11
14	200	pipeline	0,18u	nov-11

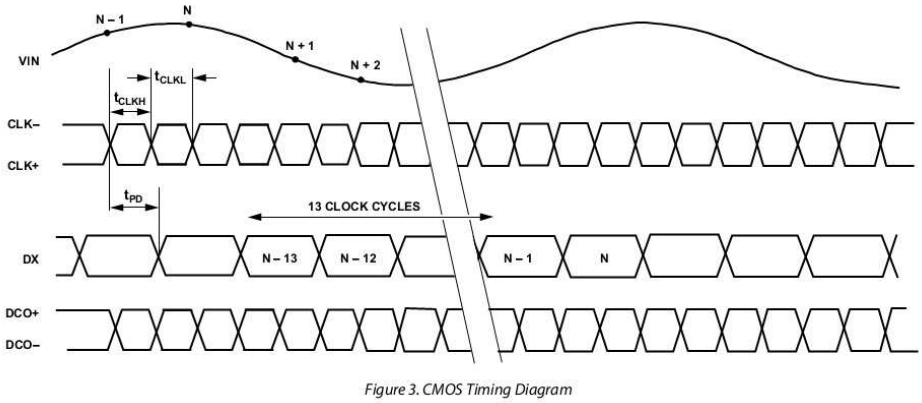

Figure 3. CMOS Timing Diagram

Figura 6.1: Diagrama de tiempos del ADC comercial. Se puede ver como el reloj de salida proporciona el sincronismo. Se observa el tiempo de latencia propio del dispositivo. La salida es la palabra digital de 14 bits.

la señal de sincronismo: DCO+. Cuando esta señal sube indica que el dato digital es válido. Si bien se puede observar un tiempo de latencia del dispositivo ADC, este no influye de manera directa en el diseño. Esta latencia tendrá un efecto cuando se utilice el ADC y la función PWL como parte de un mecanismo o sistema más complejo.

A fin de agilizar las simulaciones e implementación se optó por simular el comportamiento de un ADC. Para esto, se diseñó un bloque que tuviera 3 salidas digitales y una señal de sincronismo para esas salidas. Los valores digitales están grabados dentro de este bloque que simula un ADC y los valores digitales se repiten cada vez que comienza un test. Estos valores fueron elegidos de manera arbitraria, sólo se consideró que se encuentren dentro del rango de entrada del PWL.

Generador de reloj

El segundo bloque que se realizó fue el generador de reloj. Este bloque posee la capacidad de sincronizar con el dispositivo ADC y de obtener dos frecuencias distintas según el modo de operación del PWL. Las especificaciones del generador de reloj. según el modo de operación, son las siguientes:

- Modo FIR y/o PWL. En este caso el reloj interno tiene que tener el doble de frecuencia que el reloj proveniente del ADC. Ambos tienen que estar sincronizados con el flanco de subida con la señal DCO+.

- Modo NOE. El reloj interno tiene que producir un reloj de 6 veces la frecuencia de la señal DCO+, manteniendo su sincronización.

6.3. Arquitectura

A continuación se explica el proceso de la especificación de la arquitectura y las consideraciones de diseño. La primera cuestión para definir es la cantidad de dimensiones o variables que posee el chip, estas dimensiones están relacionas con el modelo físico o real que se quiere modelar utilizando la estructura PWL. Para el caso que se presenta se utilizaron 3 entradas para el chip.

Una vez definido el número de entradas, se obtienen los distintos modos de operación ilustrados en la Fig. 6.2. Se pueden apreciar las combinaciones PWL, FIR y NOE. Se observa que

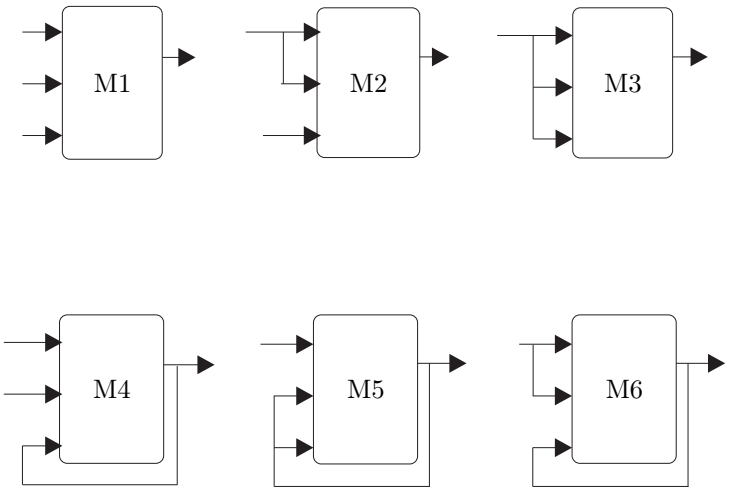

Figura 6.2: Las 6 configuraciones del chip para soportar los modos de configuración PWL, FIR o NOE. Siempre hay una entrada exterior que no se puede alterar. Los modos son: M1 es PWL 3 entradas independientes, M2 es PWL con 2 entradas independientes y una de ellas retrasada, M3 representa una estructura FIR, M4 representa una estructura NOE con dos entradas independientes, M5 es una estructura NOE con la salida realimentada y retrasada y por último M6 es una combinación de NOE y FIR

siempre existe una entrada al modelo que es independiente del modo; en este caso es la entrada: $\mathbf{u}(k)$.

Como la configuración y manejo de estas señales se realiza de manera interna en el chip, se define el núcleo que es el que realiza la función PWL y el entorno que provee la ejecución de los modos. La manera de proporcionar estas características es utilizando registros de retraso y de realimentación de la salida. Estas funcionalidades son habilitadas por un bloque de control. En el caso que se utilice el PWL con 3 entradas independientes (modo M1) estos registros quedan inhibidos. Este bloque de control, también, comanda los multiplexores que se encuentran conectados a la 2da y 3ra entrada del núcleo PWL. El bloque de control se configura durante el reset del chip, es decir, para pasar de una configuración a otra, se tiene que poner el chip en un estado de reset y luego cambiar la configuración del HW utilizando 3 pines de configuración. La Fig. 6.3 muestra los multiplexores y los registros que reciben la señal pasando por retardos unitarios.

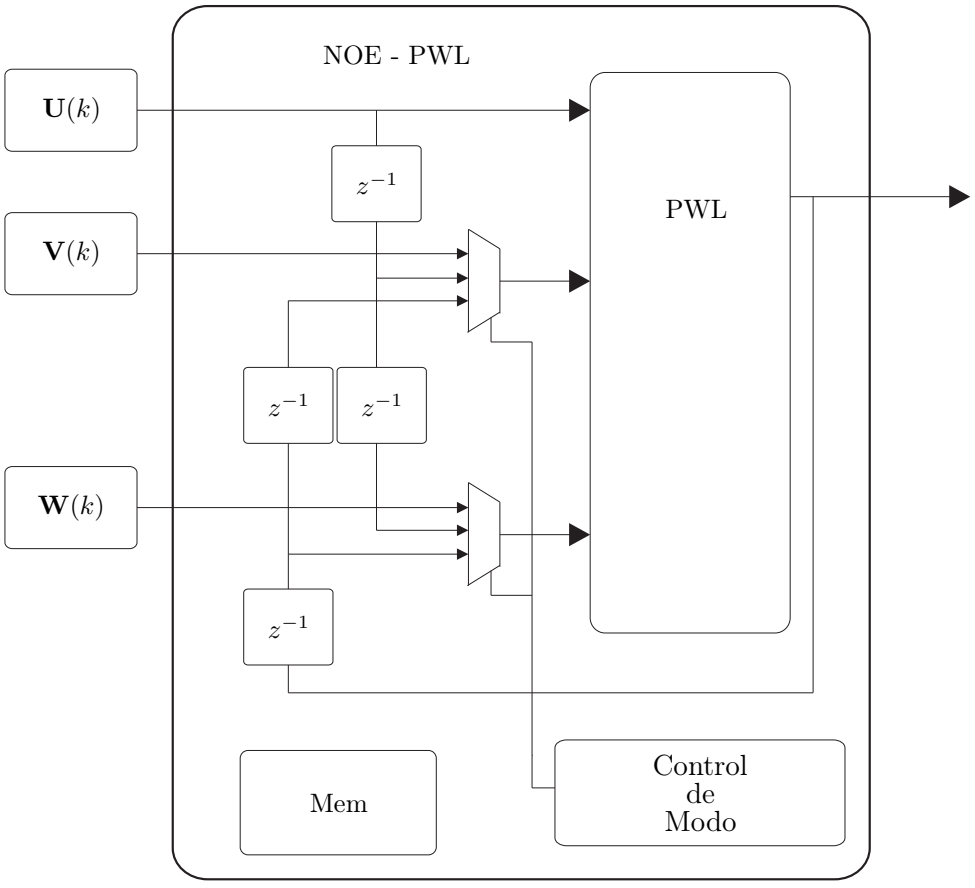

Figura 6.3: Diagrama en bloque de la estructura para representar un modelo PWL, FIR o NOE.

Una vez definido el entorno y la funcionalidad del núcleo del chip, se presentan los bloques que constituyen la arquitectura de la estructura PWL. Estos bloques se pueden ver en la Fig.

6.4. La mayoría de estos bloques son combinaciones pero su funcionamiento esta condicionado a señales de habilitación para garantizar el sincronismo. Dicho sincronismo asegura los accesos a memoria. Para el caso de los multiplicadores y sumadores se empleó una técnica conocida

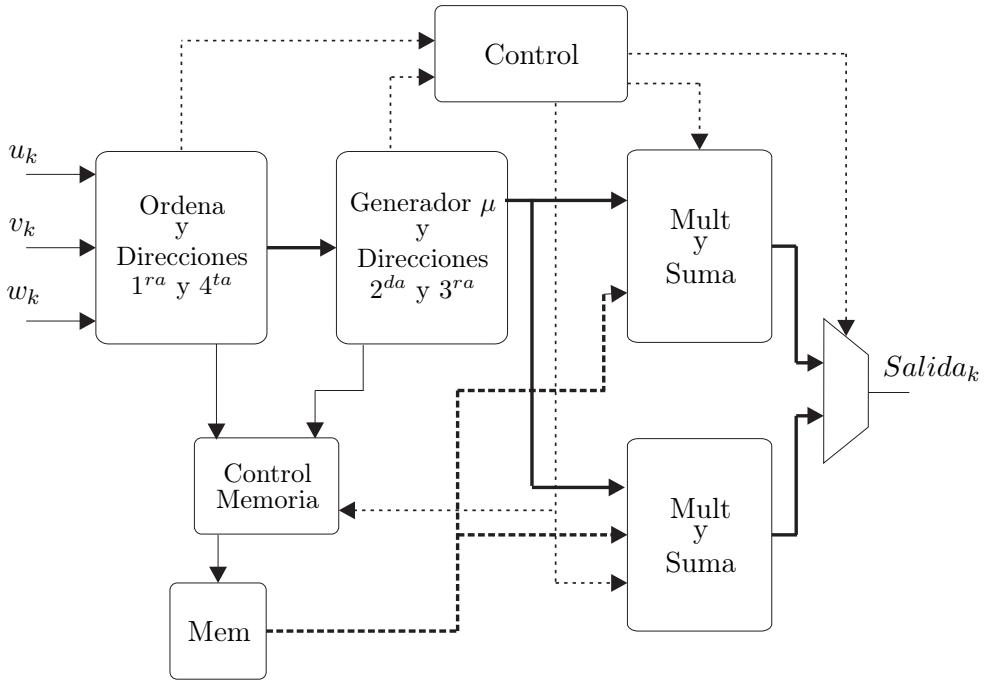

Figura 6.4: Bloques internos del chip. Contiene bloque combinacionales y sincrónicos de manera de optimizar el retardo general del sistema y aumentar la velocidad de salida de los datos procesados. Los bloque de multiplicación y suma son repetidos para poder procesar los datos que entran y su correspondiente retardo en el procesamiento.

como "tic-toc". Este nombre se debe a que ambos operan en paralelo para los distintos datos de entrada, es decir, cuando un bloque de multiplicación y suma está resolviendo para unos datos de entrada (datos par), el otro va a actuar sobre próximos datos de entrada (datos impar).

Como se dijo anteriormente, el dispositivo de memoria es un factor determinante de la velocidad del chip, por lo tanto maximizar su rendimiento es clave para el éxito de la arquitectura que se propone. La manera de optimizar el rendimiento es lograr un ciclo de lectura en cada ciclo de reloj. De esta manera, la latencia del dispositivo es despreciable cuando el chip entra en régimen de trabajo.

La forma de procesar los datos que ingresan al chip es mediante la creación de un pipeline. Una manera de resaltar los requerimientos de tiempos y las restricciones que se aplican al dispositivo de memoria fue estudiando el pipeline con 3 y 4 dimensiones. Estas estructuras se muestran en la Fig. 6.5 y Fig. 6.6, ambas utilizan un dispositivo de memoria integrado y de doble puerto de escritura y lectura. De ambas figuras se concluye que la cantidad de puertos de memoria influye dramáticamente en el resultado del rendimiento. Ambos esquemas están realizados con una memoria de 1 ciclo de reloj de latencia con dos puertos de lectura. Para el caso del R3, la

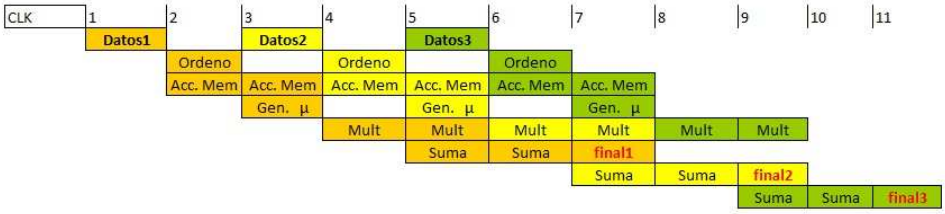

Figura 6.5: Esquemático del pipeline para un PWL de 3 dimensiones. Muestra como se accede de forma continua al dispositivo de memoria, es decir que constantemente se está leyendo de la memoria. Los distintos colores marcan los datos y su tratamiento durante la ejecución del pipeline.

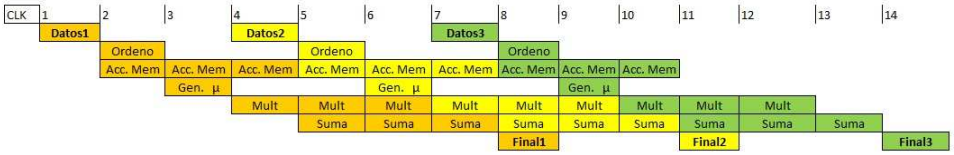

Figura 6.6: Esquemático del pipeline para un PWL de 4 dimensiones. A diferencia del pipeline anterior, se observa que se extiende la tasa de salida del PWL ya que son 5 las direcciones de memoria que se tiene leer. Al ser una memoria de doble puerto, se tiene que hacer una tercer acceso de lectura.

tasa de datos resultante es mayor que para el sistema de R4. Esto se debe a que el primero realiza 4 accesos a memoria, mientras que el segundo realiza 5 accesos. Como el dispositivo de memoria es de 2 puertos, se tiene que hacer un tercer acceso y esto agrega un ciclo de reloj al pipeline.

En base al análisis realizado y teniendo acceso a un dispositivo de memoria de doble puerto, se describe en mayor detalle la estructura para una función de 3 dimensiones. Un pipeline más detallado se explica en la Fig. 6.7.

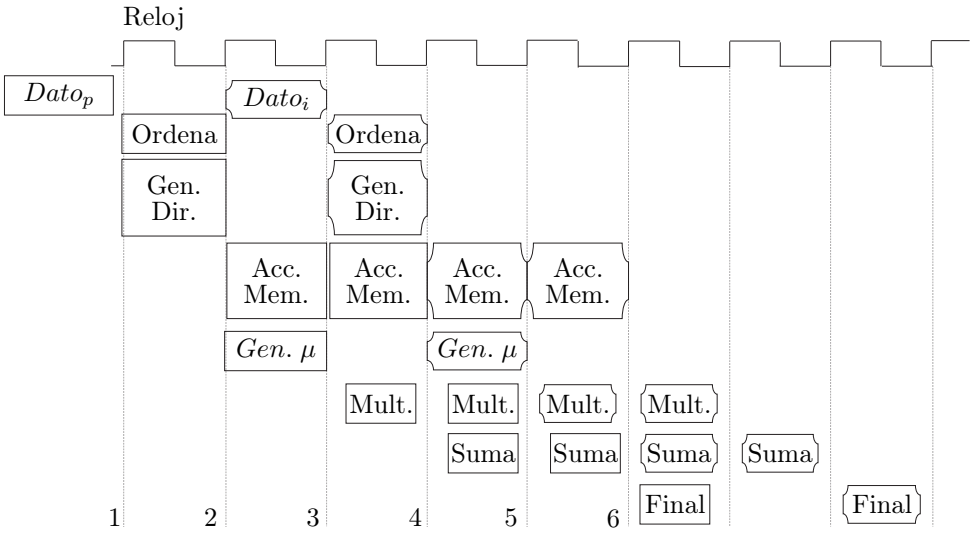

Figura 6.7: Esquema del pipeline mostrando los procesos entre reloj y los resultados de estos procesos. Nuevamente el dispositivo de memoria se utilizó en su máximo desempeño.

El detalle de las acciones que suceden es el siguiente:
Los datos son capturados en el flanco positivo del 1^{er} ciclo de reloj. En este caso el dato se refiere a las 3 variables de entrada. Entre el ciclo 1^{ro} y el 2^{do}, el bloque "Ordena"; produce la salida ordenada de los datos. En el 2^{do} ciclo reloj, el controlador de memoria busca las direcciones 1^{ra} y 4^{ta}.
En el 3^{er} ciclo reloj los valores de μ (se detalla más adelante) están listos y el controlador busca las direcciones 2^{da} y 3^{ra}. Esto completa las 4 direcciones para el primer dato de entrada. Entre el ciclo 3^{ro} y 4^{to} se realizan las dos primeras multiplicaciones, cuyos resultados se sumarán el próximo ciclo de reloj.
En el 4^{to} reloj los eventos que se producen son: la captura los datos ordenados de la entrada impar, la búsqueda de las dos direcciones en la memoria correspondiente al dato impar, el inicio de las dos segundas multiplicaciones del primer dato y la realización las sumas antes enunciadas. Entre el 5^{to} y 6^{to} ciclo se buscan las direcciones 2^{da} y 3^{ra}, del dato impar y se realiza la suma de los resultados de las últimas multiplicaciones del primer dato. Luego, el próximo flanco de reloj, el resultado PWL, está listo para salir.
Se puede notar que el dispositivo de memoria esta constantemente utilizado, como se había mencionado anteriormente. La forma de ejecución del pipeline asegura la alta tasa de salida de los datos PWL.

La funcionalidad y definición de los bloques que intervienen en el pipeline se describen a continuación. Estos tienen en cuenta la cantidad de dimensiones y características de las entradas al chip:

- Datos: Son los que provienen desde el exterior y se asegura su captura mediante las señales de sincronización. Los datos recibidos son 3 y estos vienen digitalizados en 14 bits.

- Ordena: Este bloque implementa una comparación todos contra todos, de forma combinacional. Esta comparación se conoce como "Crossbar". Esto asegura que en un ciclo de reloj los datos se ordenan de mayor a menor. Además, guarda la información del orden para poder generar las direcciones de memoria según corresponda. La complejidad de este bloque aumenta conforme aumenta la cantidad de datos, pero esta complejidad es manejable ya que siempre se puede implementar de manera combinacional, como ya se dijo.

- Acc. Mem: Corresponde a la secuencia para leer, desde la memoria, el parámetro del modelo. Como es una memoria de dos puertos, este acceso corresponde a dos direcciones de memoria diferentes. Los dos primeros accesos corresponden al principio y al final de hipercubo formado por los parámetros (vértices) del simplice. Estas dos direcciones se obtienen inmediatamente en cuanto se registraron los datos de entrada. Las otras dos direcciones se deducen del ordenamiento de los datos de entrada.

- Gen. μ: Este bloque se encarga de generar los valores denominados μ que son las diferencias de las partes fraccionarias ordenadas de mayor a menor. Se obtiene restando simplemente el primero menos el segundo y el segundo menos el tercero. El primer μ es directamente la parte fraccionaria del primero y el último μ es la unidad, menos la parte fraccionaria del dato de entrada mayor.

- Mult: Este bloque contiene los multiplicadores para hacer cada μ con el valor del parámetro de memoria. Estos multiplicadores son dos ya que son dos multiplicaciones en paralelo.

- Suma: Estos son sumadores en paralelo. El valor final corresponde a la suma de todos los μ por el valor del parámetro.

6.4. Diseño

Una vez definido el pipeline, se procede a describir la funcionalidad de los bloques que lo constituyen. La funcionalidad, en algunos casos, se extiende por más de un ciclo de reloj ya que se necesita de información previa. Este es el caso del generador de direcciones, por ejemplo, ya que dos de las direcciones son inmediatas y las otras dos direcciones necesitan información del bloque "ordena" para su cálculo final.

A continuación se describe el funcionamiento del pipeline:

Para empezar, las entradas son registradas usando la sincronización del ADC. Cada entrada tiene 14 bits, los 4 bits más significativos son la parte entera y los restantes son la parte fraccionaria. La partes fraccionarias se pasan al Bloque Ordena donde son ordenadas por comparación mediante el método de *crossbar*, como se ilustra en la Fig. 6.8. Es un bloque combinacional y el sincronizmo está dado por los registros de las entradas. Luego la variable del selector es usada para ordenar las entradas en forma ascendente, como se puede ver en el Cuadro 6.2.

Cuadro 6.2:
Selector múltiple combinacional del crossbar.

bit1	bit2	bit3	Salida
0	0	0	U V W
0	0	1	U W V
0	1	1	W U V
1	0	0	V U W
1	0	1	W U V
1	1	0	V W U
1	1	1	W V U

Luego se pasa al Bloque de Direcciones. La funcionalidad de este bloque se divide en dos: *Direcciones Base_y_Final* y *Direcciones Intermedias*.

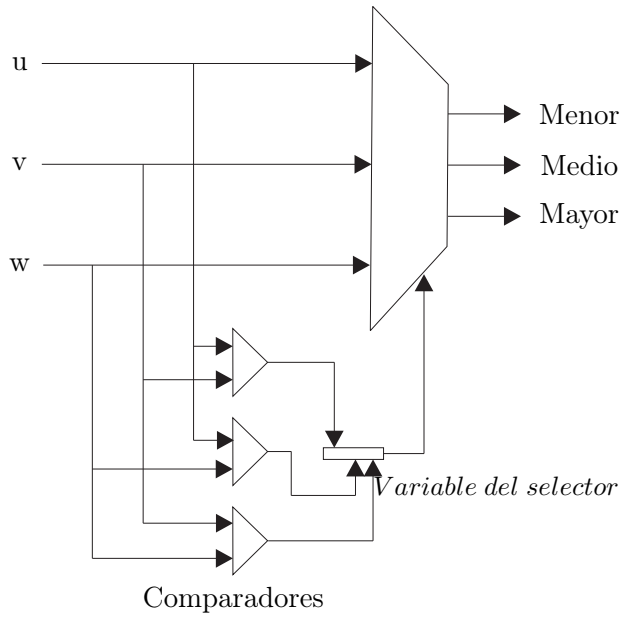

Figura 6.8: Comparaciones combinacionales del crossbar. Una vez efectuada la comparación, el resultado es utilizado para manejar el selector. Este ubica las entradas en la salida correspondiente de menor a mayor.

- Direcciones Base_y_Final: Este bloque genera las dos primeras direcciones de memoria. La dirección "base" es generada concatenando la parte entera de las entradas en el orden original. La dirección "final" se obtiene sumando 1 a cada variable.
  Ej: U_E = C, V_E = 2 y W_E = 5 la dirección base es "C25" y la dirección final será: "D36".

- Direcciones intermedias: El mismo bloque Ordena indica, según la comparación realizada, como se incrementa la dirección base de manera de obtener las direcciones intermedias hasta llegar a la dirección final. La información viene en dos variables las cuales se le suman a la dirección base:
  Ejemplo: La dirección base "C25" puede expresarse de la siguiente manera; C25 + 010 = C35 y C25 + 011 = C36 donde 010 y 011 son los valores provenientes del bloque Ordena. Así se generan las 4 direcciones que se necesitan: C25, C35, C36 y D36. Estas direcciones serán la secuencia de direcciones para obtener los vértices de un simplice del hipercubo.

Una vez que las partes fraccionarias están ordenadas, se puede proceder al cálculo de los coeficientes μ. Esto lo realiza el bloque *Generación de* μ. La obtención de μ se logra restando la partes fraccionarias de los datos ordenados de menor a mayor. Los μ son 4 y se obtienen de la

manera siguiente:

$$\begin{aligned}
\mu_0 &= Fraccionaria(x_1) \\
\mu_1 &= Fraccionaria(x_2) - Fraccionaria(x_1) \\
\mu_2 &= Fraccionaria(x_3) - Fraccionaria(x_2) \\
\mu_3 &= 1 - Fraccionaria(x_3)
\end{aligned} \tag{6.2}$$

Estas restas son combinacionales pero están sincronizadas de manera tal de comenzar cuando los resultados del ordenamiento ya está registrado.

El proceso de multiplicación es realizado por el bloque *Multiplicación*. La multiplicación opera sobre los parámetros de memoria y los μ previamente calculados. Para este bloque se utilizó la técnica, antes mencionada, denominada Tic-Toc. Esta técnica duplica el HW de manera de poder operar los datos en un pipeline. Es decir, los datos pares son procesados por un par de multiplicadores y los datos impares son procesados por otro par distinto de multiplicadores, ilustrado en la Fig. 6.9. Esto tiene la desventaja de duplicar el HW existente, pero permite la operación pipeline cuando el proceso de los datos se extiende por más de un ciclo de reloj. Para

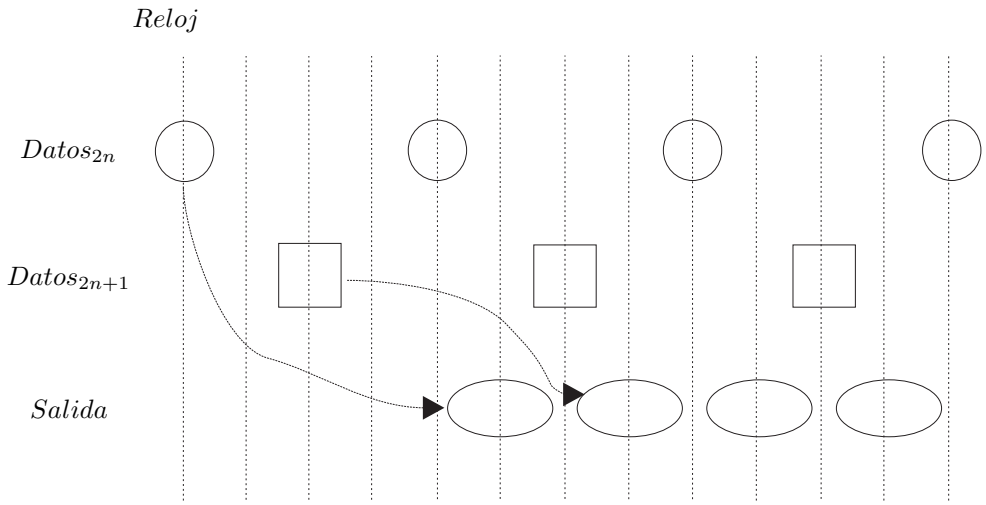

Figura 6.9: Gráfica de la implementación Tic-Toc de los multiplicadores y sumadores. Los círculos representan un cadena de procesamiento donde intervienen multiplicadores y sumadores. Este mismo hardware está contenido en los cuadrados. La operación de los círculos y cuadrados es alternado. La salida, representada por elipses, obtiene el resultado alternando entre estas cadenas de procesamiento.

el caso donde se este ejecutando en el modo NOE, solamente unos de estos conjuntos es el que procesa y el otro conjunto queda inhabilitado. Esto se debe a que en este modo las operaciones sobre los datos de entrada no son en paralelo porque se tiene que cumplir la latencia propia.

Finalmente, el bloque *Sumas* es muy similar, en su operación, al bloque de las multiplicaciones. En este caso el sumador está duplicado y cada conjunto de multiplicadores tiene su propio sumador. Para el caso del modo NOE, un par de sumadores queda inhabilitado.

6.5. Simulación e implementación FPGA

Simulaciones

En esta sección se muestran los 6 modos de operación mencionados previamente incluyendo resultados de simulaciones y comparaciones entre VHDL y Matlab. Además, se incluyen resultados de la implementación en FPGA.

El primer resultado de simulación, ilustrado en la Fig. 6.10, muestra los 6 modos de funcionamiento. Se puede ver cómo cambia el flujo de datos entre los modos FIR y NOE debido al retardo propio del chip por la realimentación que exige la estructura. Los números: 1, 2, 3...6, en la señal "Conf", son los modos. Cada uno de ellos ejecuta una serie de datos de entrada.

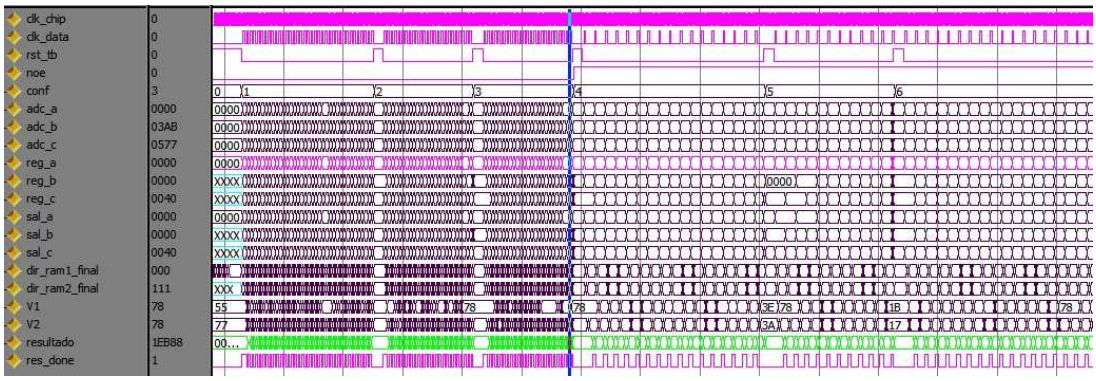

Figura 6.10: Simulación donde se puede ver los 6 modos de funcionamiento que tiene que soportar el diseño del nuevo chip. El chip se encuentra en estado de reset para el cambio de configuración. En los modos que hay realimentación de la salida, la tasa de datos baja notablemente por el retardo propio del chip.

A continuación la Fig. 6.11 muestra cómo las entradas del PWL son la entrada actual y el retardo de las entradas anteriores. El resultado está listo en el ciclo de reloj número 6. Sobre el margen izquierdo las señales denominadas: reg_a, reg_b y reg_c corresponden a la entrada del modelo PWL. Esto es para representar un modelo FIR. Se observa que la tasa de datos de salida es la frecuencia de operación dividido 2 y la latencia son 6 ciclos de reloj. Seguidamente, se muestra el modo tipo NOE en la Fig. 6.12. Donde se puede observar que contiene la entrada original que proviene del ADC y las otras dos entradas corresponden a las salidas del modelo PWL anteriores. Se puede notar que la tasa de datos se redujo pero se mantuvo el retardo de salida.

La manera de comprobar el diseño es mediante la comparación directa de los resultados de simulación con los resultados de un modelo de alto nivel realizado en Matlab. Es interesante destacar que los efectos de cuantizaciones estudiados en los capítulos anteriores no se observan, ya que los datos utilizados en el Matlab provienen del simulador digital. En otras palabras, el

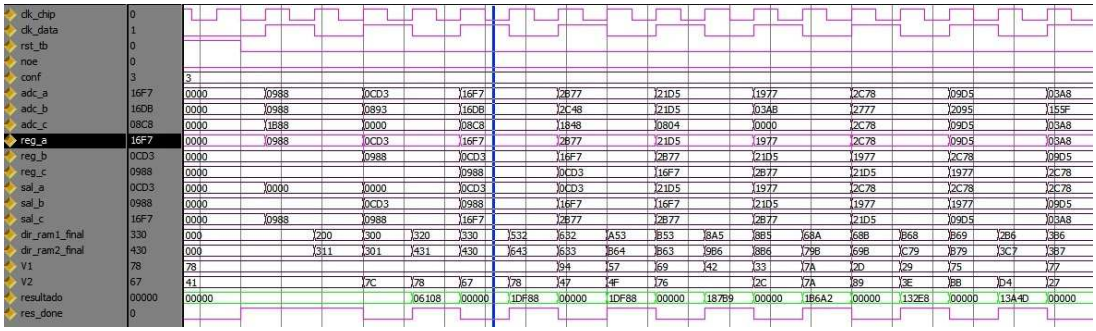

Figura 6.11: Salida para el modo número 3 de operación. La entrada del PWL es la entrada 1 y dos muestras retrasadas de la entrada. El resultado se obtiene 6 ciclos de reloj después.

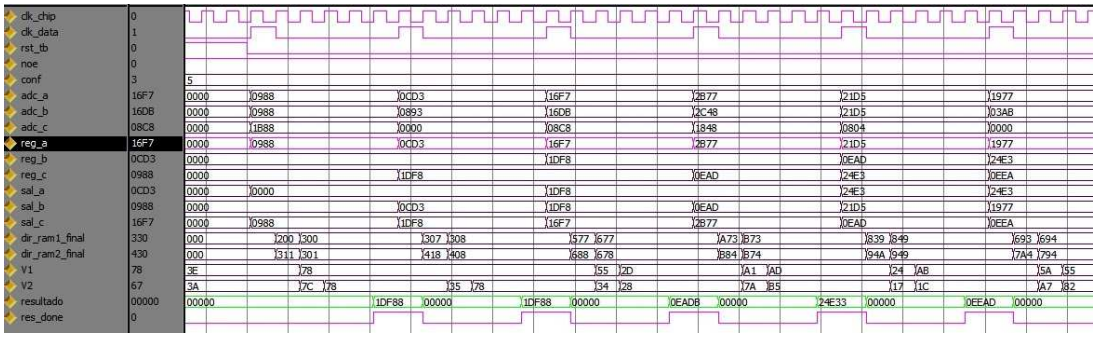

Figura 6.12: Salida del chip para el modo 5 de funcionamiento. Claramente se ve como el sistema tiene que esperar el resultado para poder ingresarlo nuevamente a la entrada. La tasa de datos se redujo pero se mantiene el retardo del sistema.

agregado de más bits de cuantización y punto flotante de la PC, no agrega más información que la disponible en los datos digitales originales del simulador. La Figura 6.13 muestra el funcionamiento tipo PWL y tipo FIR.

La Figura 6.14 muestra el resultado de la comparación entre el simulador y el modelo del Matlab para el funcionamiento tipo NOE. Como se mencionó en los capítulos anteriores, en este caso también existe un truncamiento de la salida de la estructura NOE para poder ingresar este dato a la entrada. La salida está cuantizada en 18 bits y la entrada solamente tiene 14 bits. El recorte se produce en los últimos 4 bits menos significativos.

Implementación

- Inicialización de la memoria del Chip: De manera de poder comparar los resultados de la implementación con las simulaciones, se debió inicializar la memoria del chip con valores preestablecidos. Para esto se organizaron los valores de los parámetros de la estructura PWL en un archivo de texto y se utilizó la herramienta Hyperterminal de Windows para enviar los valores por el puerto RS232 de la PC. Dentro de la estructura se realizó un mó-

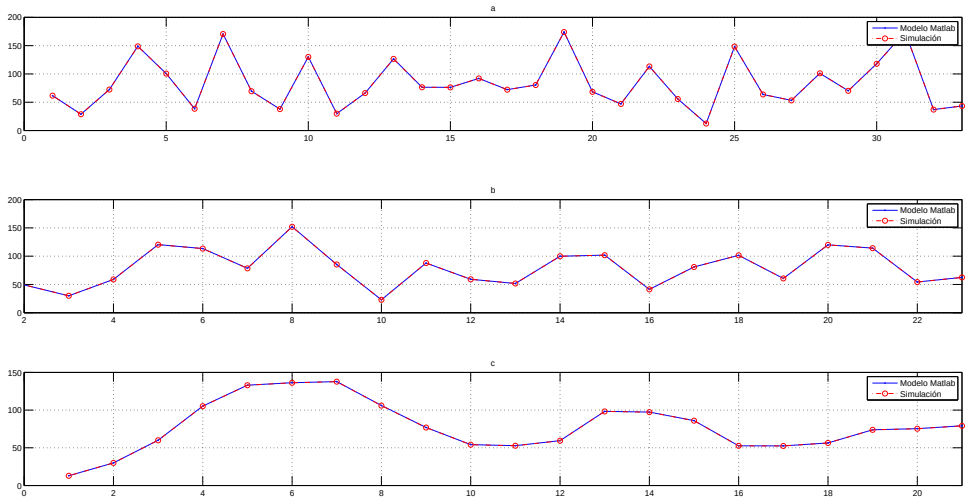

Figura 6.13: Resultado de la comparación entre el simulador y el modelo realizado en Matlab. Las comparaciones se realizó para los 3 primeros modos: a) Entradas independientes (M1). b) Dos entradas independientes y una retrasada (M2). c) Una entrada independiente y dos entradas retrasadas (M3). Se puede apreciar que los resultados son exactamente iguales.

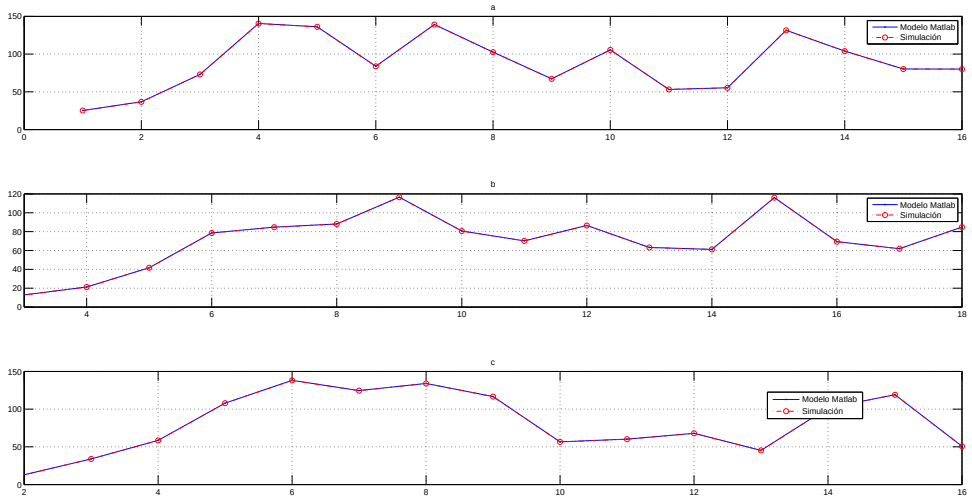

Figura 6.14: Resultado de la comparación entre simulador y el modelo realizado en Matlab. Las comparaciones se realizó para los 3 últimos modos: a) Dos entradas independientes y una salida realimentada (M4). b) Una entrada independiente y dos salidas realimentadas (5). c) Una entrada independiente, una entradas retrasada y una salida realimentada (M6).

dulo que obtenía el valor del puerto RS232 y lo escribía en la memoria. Durante la escritura de la memoria, el chip se mantenía en estado de reset. Los valores de dicha memoria se

obtuvieron de los archivos de simulación y se ordenaron los datos de manera de respetar el orden creciente de las direcciones de memoria. La cantidad de datos a escribir fue de 4096 Bytes.

- Validación del diseño: Para facilitar la rápida comparación de los datos, en todos los modos de operación, y comprobar la exactitud del diseño se agregó una estructura FIFO interna para almacenar los resultados. La secuencia de datos comienza con la señal de reset y se sincroniza con la señal "done" que marca un dato válido de salida. Luego, mediante el PicoBlaze esos datos capturados son enviados a la PC mediante la RS232. De esta manera se compara el resultado de la FPGA con el obtenido en matlab o la simulación en Modelsim. Para poder configurar los modos de trabajo se utilizó un dip switch provisto en la placa de desarrollo. Estos modos se configuran manteniendo el PWL con la señal de reset habilitada. El tamaño de la FIFO es de 80 valores y una vez alcanzado este tope se desabilita para luego ser leídos con el puerto serie.

- Validación de las especificaciones: A los efectos de ilustrar el resultado del desempeño del chip en cuanto a las especificaciones que se querían satisfacer, se utilizó el adquisidor digital Lecroy para capturar la salida de datos de la FPGA. Para esto se modificó ligeramente el diseño de manera de poder sacar hacia los pines de salida las señales de: reloj, "done" y el resultado de la función PWL. Estas señales son mostradas en la Fig. 6.15. La señal "done" que indica cuando el dato de salida es válido, la señal "rst" es el reset de la FPGA y el bus "datos" que son los bits del resultado. La Fig. 6.16 muestra la salida del simulador donde se marcaron los datos mostrando la coincidencia con lo obtenido en el analizador lógico. Los demás datos que aparecen en el analizador lógico corresponden a falsos muestreos del analizador ya que éste muestrea a una frecuencia fija de 1GHz. Generalmente se usa el modo "state machine" en los analizadores lógicos pero esta funcionalidad no está disponible. Además, se comprobó el problema de "signal integrity" en la FPGA al tener las salidas sin terminación y a 50MHz. El analizador capturó todas las variaciones de las señales entre 0 y 3,3 volt. Las Figs 6.17 y 6.18 muestran la secuencia de datos de salida para el modo NOE en el analizador lógico y la correspondiente salida en el simulador. Se puede ver como efectivamente la tasa de datos se reduce. En cuanto a las capturas falsas, también están presentes en este modo pero se puede ver como el efecto se ve minimizado por la reducción efectiva de la frecuencia de salida de los datos por estar trabajando en el modo NOE. En este caso la frecuencia de salida es de 16MHz en comparación con los 50MHz del PWL.

  La arquitectura diseñada maximiza el uso del dispositivo de memoria haciendo que se encuentre en constante actividad. De hecho, esta arquitectura presenta dos fases de accesos a memoria: la primera fase consta de: dirección Base y dirección Final. Luego del próximo ciclo de reloj, después de la ordenación, la segunda fase permite obtener todas las direc-

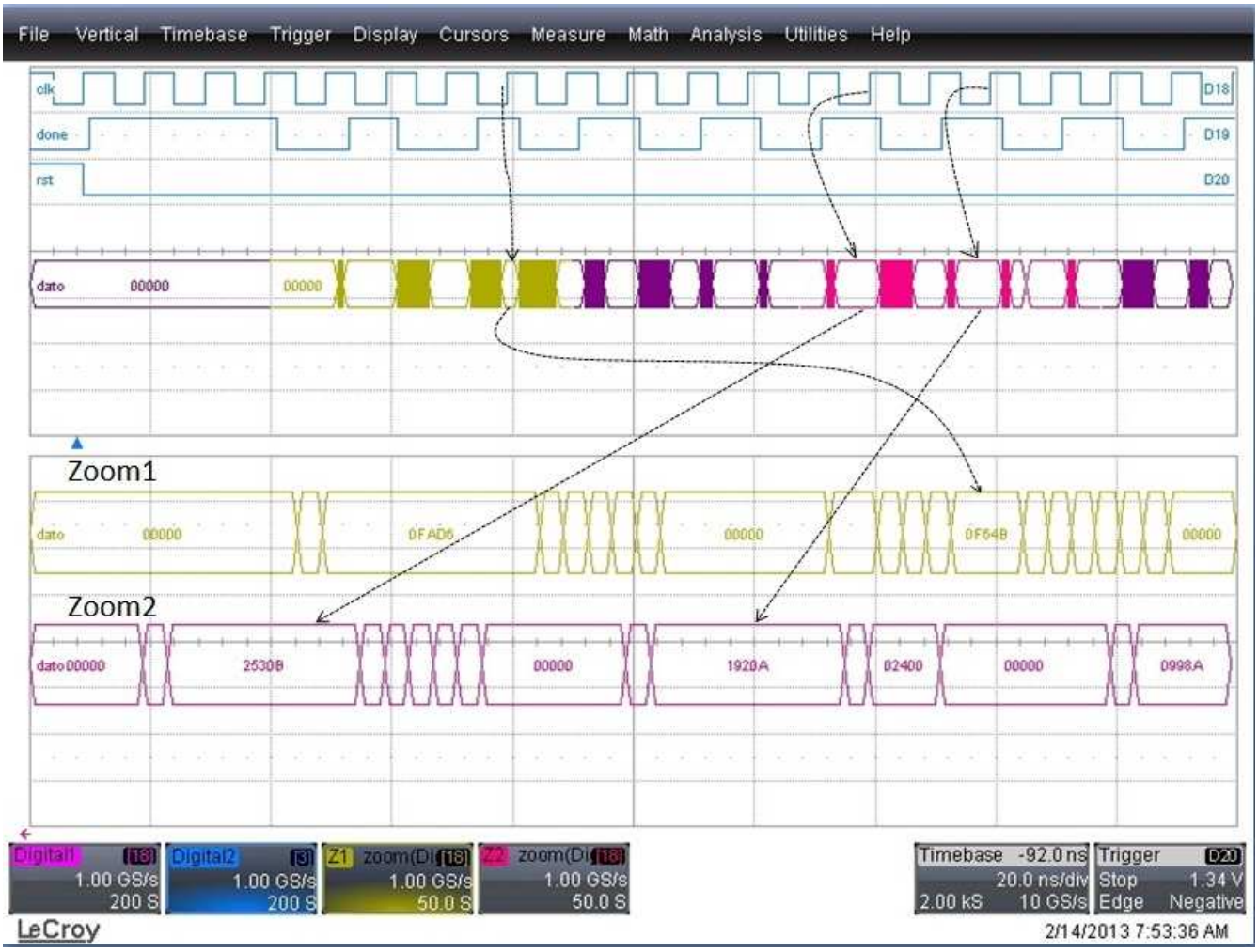

Figura 6.15: Captura de la salida del PWL en modo FIR. Se observan 2 "zoom" de dos partes de la salida de manera de marcar los datos relevante. Se puede ve que la frecuencia de salida es la mitad del reloj de trabajo. La otra actividad que se ve en la figura corresponde a falsedades ya que el analizador no se puede sincronizar con el chip y la frecuencia de muestreo corresponde a 1GHz.

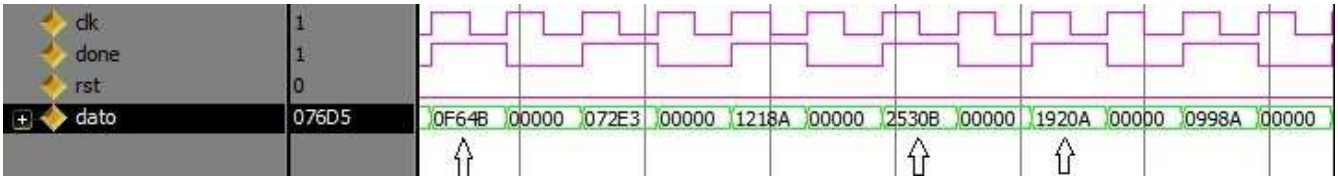

Figura 6.16: Salida del simulador donde se muestran los datos de salida coincidentes con la captura del analizador lógico.

ciones que conforman el hipercubo. Llegado el caso de tener que implementar un PWL de orden mayor se requerirá de un dispositivo de memoria multi-puerto para permitir la búsqueda de las direcciones en forma paralela sin tener que influir en el tiempo de retardo. Es necesario aclarar que se tendrán que agregar los multiplicadores y sumadores necesarios

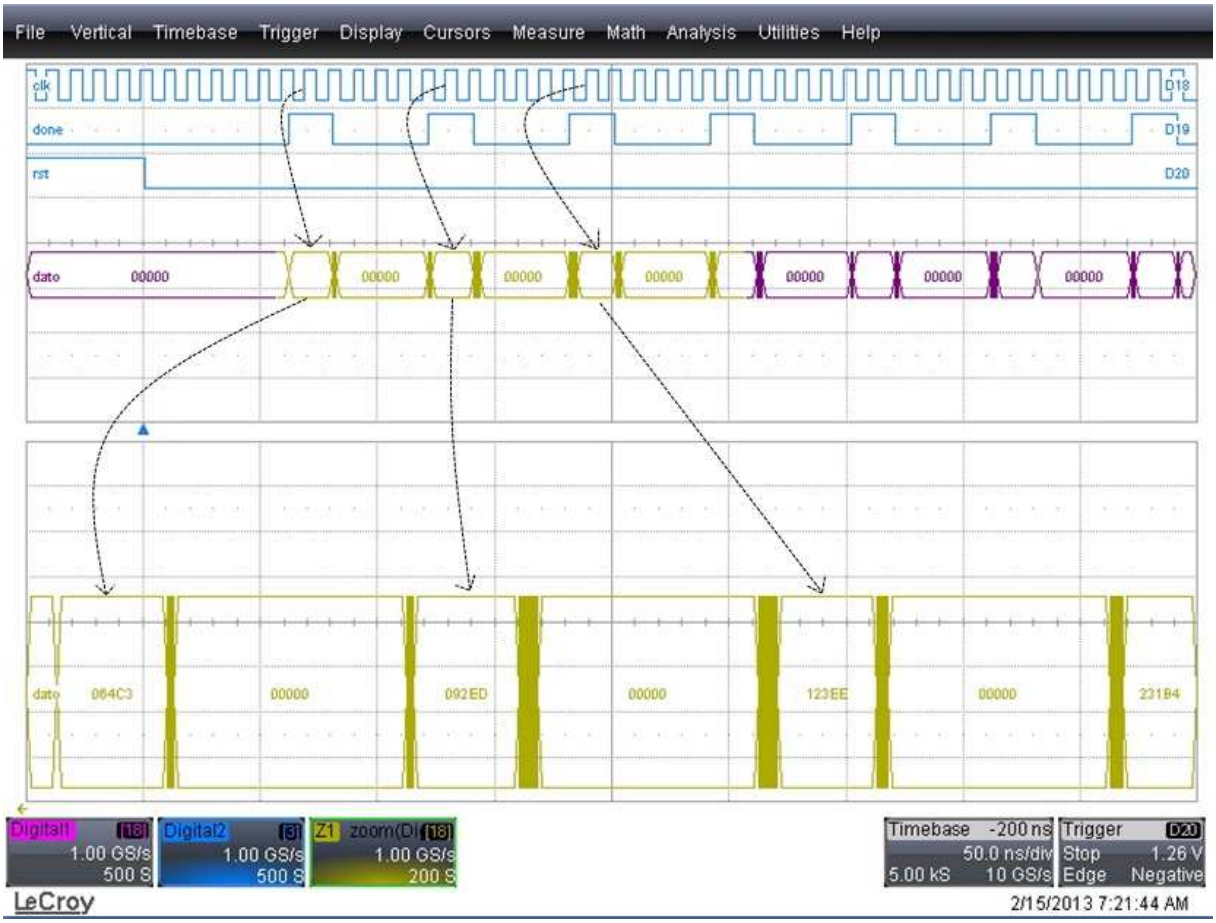

Figura 6.17: Captura del analizador lógico en modo NOE. Se observa como la tasa de datos de salida es inferior en comparación con el modo PWL debido al tiempo de retardo del chip.

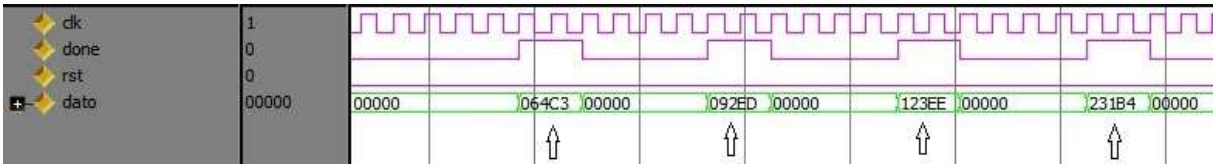

Figura 6.18: Captura del simulador. Se puede observar la correspondencia de los datos de salida con lo obtenido en la FPGA.

para cumplimentar el cálculo PWL. De esta manera se sigue manteniendo la tasa de datos y el retardo del sistema, a expensas de este agregado.

6.6. Conclusiones

Se demostró el impacto directo del dispositivo de memoria para la implementación de una estructura PWL cuando el requerimiento principal es la velocidad de cómputo. Para obtener una

implementación veloz, la memoria tiene que ser integrada. Sin embargo, esto es perjudicial para la precisión de la aproximación, dado que dentro del chip, el tamaño para la implementación de la memoria está más acotado. Esto se puso en evidencia mediante la realización de arquitecturas tipo pipeline donde se vio que el tiempo de respuesta (latencia) es un factor crítico en el desempeño. Además, se vio cómo la característica multi-puerto del dispositivo de memoria juega un papel importante en el resultado de desempeño logrado.

Se realizó, implementó y verificó una estructura PWL que cumple con los factores de desempeño requeridos, como ser: velocidad de computo y tiempo de latencia. Además, se agregó un circuito adicional de manera de poder efectuar distintas configuraciones con el mismo chip. Estas configuraciones se determinan previamente durante el reset del chip.

Capítulo 7

Conclusiones Generales y Trabajo Futuro

En la presente tesis se han abordado todos los aspectos centrales de la implementación, en un chip, de una técnica de identificación de sistemas dinámicos no lineales. Esta establece un camino de abordaje de los principales aspectos a tener en cuenta desde la estructuración del modelo a utilizar hasta su implementación en hardware.

En el Capitulo 2 se introdujeron los conceptos básicos de las bases utilizadas en las estructuras PWL. Se expusieron las nociones de las estructuras NOE y FIR. Para finalizar, se presentó las arquitectura del chip realizado en el grupo de investigación que fue el punto de partida para los trabajos realizados.

En el Capítulo 3 se describió la estructura PWL utilizada para el modelo NOE. Se presentaron los errores propios de la digitalización y la truncación propia de la estructura NOE, este análisis se realizó en ejemplos realizados sobre un ASIC. Además, se analizaron dos posibles métodos de identificación marcando sus ventajas y desventajas, sobre todo, los beneficios en ambientes ruidosos. Se expusieron las necesidades de velocidad de cálculo y latencia del sistema para poder implementar este tipo de estructuras. Se puso en evidencia el compromiso entre el tamaño de la memoria y su latencia, siendo este compromiso importante en el momento del diseño de la estructura digital.

El Capítulo 4 presentó una estructura FIR utilizada para la corrección de un ADC comercial. Se introdujeron los parámetros característicos de estos dispositivos y se implementó un conjunto de medición y caracterización en el laboratorio incluyendo filtros para cumplir con los niveles de relación señal a ruido. Se mostró una posible arquitectura para soportar esta implementación. Se caracterizó en dispositivo real y se vio cuales son las figuras de mérito que marcan a estos tipos de dispositivos. Se demostró la complejidad de la identificación de estos tipos de dispositivos para la compensación. Se planteó un compromiso entre el HW necesario para la corrección y el resultado de esta compensación en términos de bits ganados en el ENOB. En términos generales, se trabajó

con aproximaciones en ambas bases mostrando sus ventajas y desventajas usando lo mejor de cada una. Se da como conclusión: una propuesta de precisión necesaria para el sistema digital y la clara necesidad del tipo de señales de entrenamiento: SINAD suficiente y buena cobertura del espacio de entrada.

En el Capítulo 5, se presentó una arquitectura de un PWL de propósito general y las pautas de diseño, si se quiere acotar el error producido entre la función multi variable original y su aproximación PWL. Se hizo un análisis de los tamaños de memoria ya que eran una parte importante del diseño dado que influyen fuertemente en el área del chip y en el desempeño del mismo. Se planteó el impacto que las variables de diseño tienen en la arquitectura y el tamaño del dispositivo de memoria. Se demostró el fuerte impacto que tiene el tamaño de la memoria en la precisión del resultado obtenido en la aproximación

El Capítulo 6 desarrolló un nuevo diseño de un PWL que mejora ciertos aspectos del chip anterior en cuanto a frecuencias de operación en los modos de PWL, FIR y NOE. Se diseño un pipeline que maximiza la utilización del dispositivo de memoria. Se presentó una arquitectura donde se utilizó un ADC comercial como ejemplo para digitalizar las entradas a este chip. Luego se ilustraron las simulaciones de los 6 modos de operación y se expuso el resultado de su implementación en FPGA para dos de los modos. Se obtuvo una arquitectura que maximiza el rendimiento y minimiza la latencia a expensas de una memoria integrada y multi-puerto. Se demostró la vinculación entre el rendimiento y la latencia del dispositivo de memoria. Se arribó a la conclusión que es necesaria la utilización de una memoria multi-puerto para lograr los objetivos respecto de la taza de salida de resultados.

Propuesta de trabajo futuro

Uno de los trabajos futuros que surgen del desarrollo de la presente tesis es la mejora del algoritmos de identificación haciendo más eficiente el cálculo del gradiente. Esto podría ser utilizando las bondades de la base estándar para luego pasar a la base general que permite su implementación digital. En este aspecto, sería de suma utilidad lograr el algoritmo de ajuste de la base estándar aplicado a la base general. Esto permitiría la mejora de la precisión utilizando lo obtenido en la identificación anterior de menor precisión. En el referente al vínculo entre la velocidad y tamaño de la memoria, se tendría que proponer un algoritmo para poder utilizar, de forma eficiente, las bondades de una memoria "cache". Los controladores de memoria comerciales poseen este tipo de manejo de manera integrada.

El avance en un caso concreto donde la utilización de una estructura NOE en el contexto del Control Predictivo basado en modelo Model Predictive Control (MPC) podría mostrar, claramente, la ventaja de tener una plataforma como la desarrollada en la presente tesis.

Con respecto a la corrección de un ADC se tiene que investigar el caso puntual donde la ganancia obtenida al aumentar el parámetro ENOB de un dispositivo, justifique la inversión de HW necesario para lograrlo.

Apéndice A

A.1. Introducción

Este apéndice se describe la validación realizada a una implementación PWL en un ASIC, que se diseñó en el grupo de trabajo. Describe las técnicas de Design For Testability (DFT) que se integraron junto con el diseño para garantizar la ejecución de la validación y chequeo de forma óptima. Además, se detalla el entorno del controlador del ASIC realizado en una FPGA y se explicitan datos sobre las técnicas de verificación formal y de validación masiva. Se explica el desarrollo de métodos de medición de consumos y de estimación de máximas frecuencias de operación.

La primera parte explica la arquitectura del ASIC para luego exhibir el entorno que permite el funcionamiento del chip y que permitió su validación en laboratorio. Seguido, se presentan las herramientas que se introdujeron en el diseño ASIC para obtener visibilidad y control en el chip. Debido a que el chip utiliza memoria externa, se procede a la explicación del manejo de memoria y como se obtiene las direcciones durante su ejecución. A su vez, se explica cómo se obtuvo el controlador y el adaptador de protocolos que se diseño de manera de conectar el chip con el controlador de memoria que se implementó en la FPGA. Se expone, luego, los resultados del proceso de validación, consumo y estimación de la frecuencia de operación, para el final, se presentan las conclusiones pertinentes al capítulo.

A.2. Breve Descripción del Chip

La Figura A.1[1] presenta, de forma esquemática, la arquitectura del chip que fue implementado utilizando un flujo de diseño EDA basado en las herramientas de Synopsys utilizando la biblioteca estándar de celdas AMI 05 OSU [5]. El chip fue diseñado para ejecutar el cálculo de una función PWL de R6 con un alto grado de la flexibilidad [6]. Una unidad de control microprogramada permite el arreglo de configuraciones diferentes utilizando el Instruction Set Architecture (ISA) del chip. Saltos absolutos y relativos, Arithmetic Logic Unit (ALU), escritura y lectura de

[1] Cortesía del Dr. Agustín Rodríguez

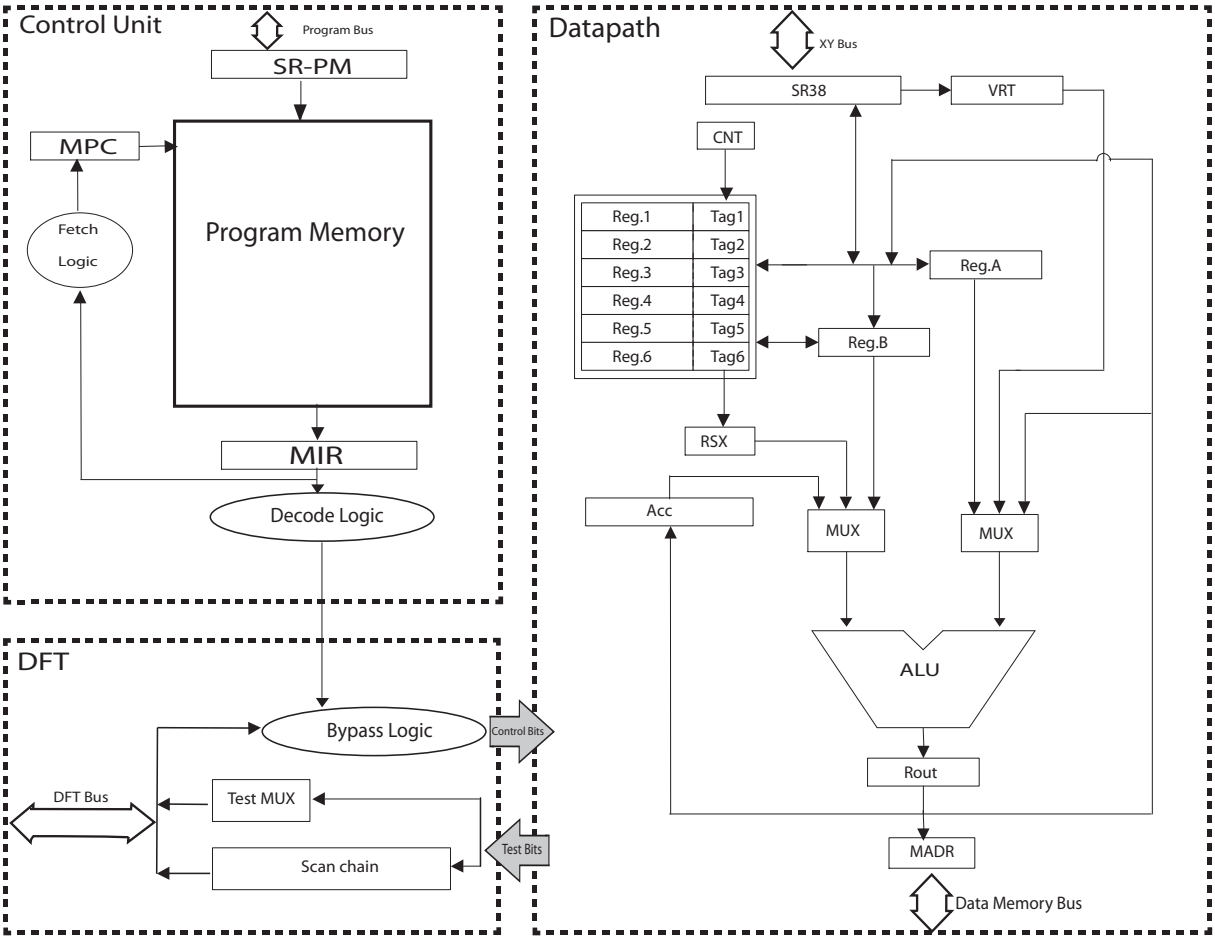

Figura A.1: Esquemático de la arquitectura del ASIC. Representación de los bloques funcionales del chip.

memoria e instrucciones de acceso a registros, proporcionan un ambiente óptimo para explotar las funcionalidades del chip.

En la Fig. A.1 se distingue un bloque denominado “Program Memory” que es la memoria interna del chip. Ésta contiene el programa que ejecuta este microprocesador orientado, programa que se encarga de recibir los datos de entrada, ordenarlos, generar los indices de memoria externos y hacer el cálculo PWL correspondiente una vez obtenidos los parámetros del modelo.

Otro bloque que se observa es el denominado DFT, éste es un bloque especial que se diseñó para aportar las herramientas para la etapa de validación una vez obtenido el chip luego de su fabricación. Esta etapa se conoce como "post-silicon” y consiste en la validación del diseño.

A.3. Entorno de Funcionamiento

El entorno que contiene al chip PWL está compuesto por dos partes: el SW y su parte de HW.

La conexión de este chip con el mundo exterior se puede observar en la Fig. A.2

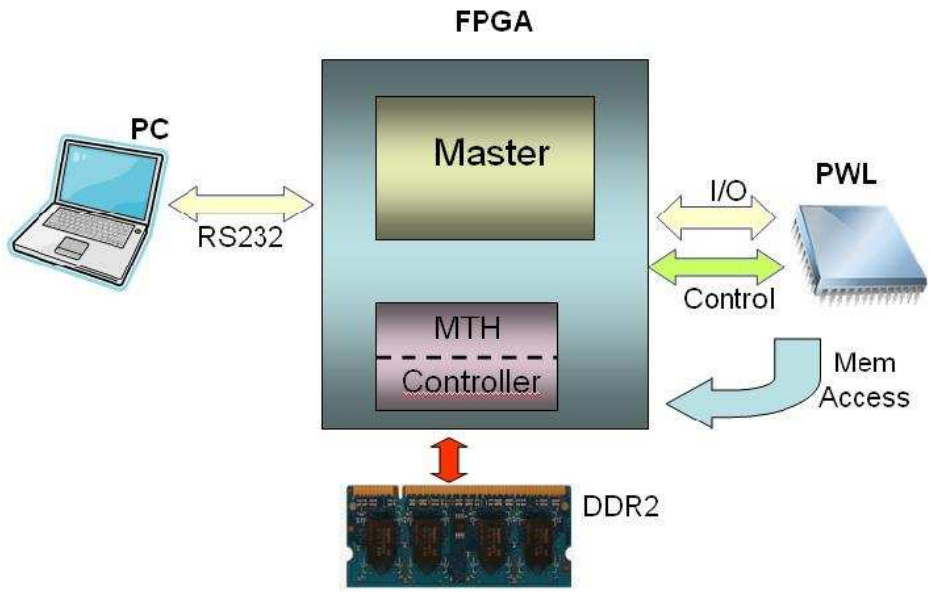

Figura A.2: Entorno del conexionado del chip PWLR6 con el mundo exterior.

La parte de HW consiste en una configuración maestro-esclavo donde el chip está en el papel de esclavo y el maestro se desarrolla en un plataforma de Xilinx con una FPGA Virtex5. El chip utiliza un dispositivo de memoria DDR. Además, posee un controlador de memoria y un Adaptador de Protocolos (AdP), que se realizaron en la Virtex5. El AdP es responsable de adaptar el protocolo del controlador de memoria DDR con el protocolo de memoria que implementa el chip. Además, este AdP, permite la funcionalidad de éstos sistemas en diferentes frecuencias: Maestro@100MHz, controlador de memoria @160MHz y PWLR6 @ [6,25 MHz - 50 MHz]. El maestro se interconecta con una PC, a través de RS232, usando Matlab como interfase.

El chip es soportado por un Printed Circuit Board (PCB) de dos capas. Este PCB incluye, además, algunos conectores para el analizador lógico y osciloscopio para facilitar mediciones de corriente y tensión durante la validación. La alimentación del chip se hace utilizando una fuente externa variable de manera de permitir variaciones de tensión durante la validación.

En la parte de SW, existen varios niveles en el sistema, cada uno de ellos se ejecutan en una parte diferente del entorno de HW:

- SW del esclavo (Chip): Este código corre en el chip y posee varias funcionalidades: recibir los datos de entrada de N-dimensiones (Xi) que son enviados por el maestro, ordena los datos de entrada para conocer las direcciones de memoria, calcula o evalúa la función PWL y envía los resultados de regreso al maestro.

  Demás está decir que este chip puede ejecutar cualquier acción por micro-programado, por

eso, otros códigos se desarrollaron para propósitos de validación y medición del consumo. (Esto se comenta, más adelante, en la parte de DFT).

El chip admite 256 palabras de código de programa de 20 bits que son alojadas en una Read Only Memory (ROM) de instrucciones. Los códigos de "assembler" son programados en esta ROM por el maestro cuando el chip se encuentra en modo de programación. Los modos de programación y ejecución son controlados por el maestro.

- SW del maestro: En el maestro se encuentran cuatro máquinas de estado y un PicoBlaze. El PicoBlaze es un microcontrolador de 8 bits definido en VHDL y provisto por Xilinx como un "Softcore". Las funciones del PicoBlaze son tres: operar la RS232 para entrada y salida de datos, activar las máquinas de estado en función de los comandos recibidos por la RS232 desde el Matlab e inicializar la memoria externa, DDR, del PWL con los parámetros del modelo correspondientes.

  A continuación se describen las máquinas de estado:

  - Carga ASM: Programa el código del chip dentro de la ROM. Esto se realizó mediante dos señales asincrónicas: datos y reloj. Ambas señales son generadas por el maestro.

  - Chip_Clk_Gen: Se encarga de generar el reloj para el chip. Esta máquina se encarga de detener y activar el reloj (Stop_Clock y Do_1_clk). (Véase la parte DFT). Además, es la que permite subir y bajar la frecuencia de operación del chip.

  - Exe_Calc_Func: Activa el chip. Esta máquina pone el chip a trabajar mediante el envío de las entradas en serie de las variables X_i. Luego espera a que el chip envíe el resultado de regreso. Esta maquina enviará los resultados finales a la interfaz del Matlab para su presentación y formateo correspondiente.

  - DFT_block: Genera la activación y control de los bloques de validación y observabilidad: Scan, BypassScan y Bypass. (Véase la parte DFT). Esta máquina es controlada por el PicoBlaze dependiendo de los comandos enviados desde interfaz humana. Esta máquina, también, realiza el formateo de datos antes de llegar a la interfase de la PC.

- SW de la PC. Es una interfase de Matlab que se ocupa del envío de comandos al PicoBlaze y el formateo de los datos para la interfase humana. Además, crea los casos al azar y compara los resultados del chip y de una función PWL que se elaboró en Matlab.

A.4. Introducción de los DFT

A continuación se describen las estructuras para validación y testeo. La idea principal de las estructuras de testeo es la de colaborar con el diseñador en las tareas de validación y verificación. Estas estructuras se diseñaron durante el desarrollo del chip y su funcionalidad fue validada en simulaciones.

El chip contiene 6 estructuras:

- Stop_Clk: Algunos eventos, ya sean internos o externos, pueden hacer que el reloj del chip se detenga, logrando detener la ejecución instantáneamente. Los eventos internos pueden ser sincronizados con el simulador para una eventual correlación temporal entre el simulador y el chip. Esta sincronización permite una fácil verificación "paso a paso" a nivel de registro. La sincronización es garantizada utilizando un contador de periodos de reloj en ambas partes: en el simulador y en el chip. La opción externa para detener el reloj está contemplada en una señal de la interfase entre la PC y el chip pero la correlación temporal no esta garantizada.

- Do_1_Clk: Genera un pulso aislado de reloj en el chip. Esto permite correr el chip paso a paso. La activación de esta señal puede ser interna o externa. La aplicación interna es cuando es utilizada por otro DFT. La opción externa esta contemplada en la interfase entre PC y el chip.

Los dos DFT siguientes están relacionados con la observabilidd del chip. La idea es sacar del interior del chip las señales o estados hacia el exterior de manera de contar desde el entorno de trabajo con señales para verificación. Estas señales son elegidas del camino de datos y del camino de control.

- MUX: Este es un multiplexor interior al chip que es activado desde el exterior. Este multiplexor permite sacar hacia el exterior distintas señales internas. La salida de esas señales seda en paralelo. La relación entre el número de bits y el número de registros "observables", es una relación de compromiso y depende de consideraciones de diseño. Este DFT tiene una activación "on the fly", es decir que su activación se puede hacer con el chip en funcionamiento y las señales elegidas que son sacadas al exterior irán, eventualmente, cambiando en cada ciclo de reloj. Cabe mencionar que este es un DFT de costo elevado debido a la cantidad de I/O pines que utiliza. Por otro lado, es el DFT más sencillo de diseñar. Este DFT permite la creación de una "firma" de las señales que fácilmente son llevadas a un analizador lógico permitiendo la comparación con el simulador. Las señales de salida del MUX y su respectiva configuración está representado en la Fig. A.3.

- Scan: Este DFT saca hacia el exterior un número diferente de bits que provienen de diferentes bloques del chip. Es serial en un formato de configuración "daisy chain". En oposición

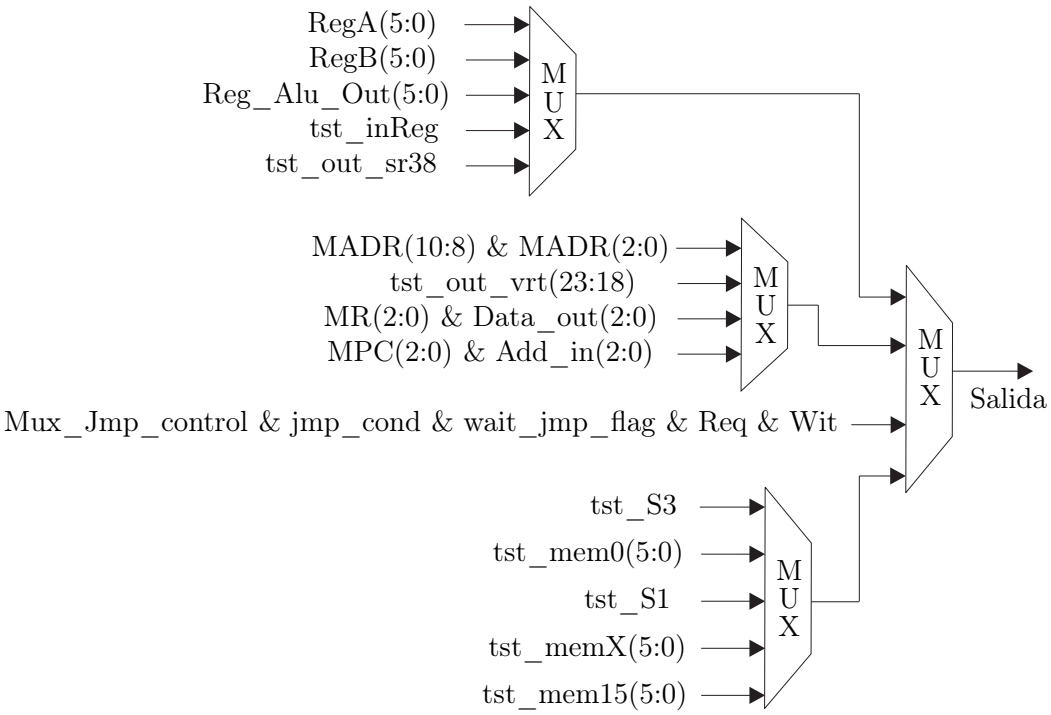

Figura A.3: Vemos la representación de las señales que entran al MUX de manera de poder sacarlas hacia el exterior del chip. Este DFT permite sacar los bits "on the fly", es decir con el chip en funcionamiento.

al MUX, el Scan tiene que ser activado cuando el chip se encuentra en un evento de "paro de reloj" para garantizar que todos los bits encadenados pertenezcan al mismo periodo de reloj. En cuanto al numero de I/O pins, este DFT es económico ya que utiliza un pin de data y otro de reloj. El reloj de este DFT es ingresado externamente y es asincrónico. En este caso es el maestro quien genera este reloj. En la implementación utilizada, el scan tiene 56 bits formados por la siguiente concatenación de bits:

control_bits (7:0) & **RegA (3:0)** & **RegB (3:0)** & **Bi_BusA (3:0)** & **Bi_BusB (3:0)** & **reg_Alu_Out (3:0)** & **reg_Acc_Out (3:0)** & **bus_out_vrt (3:0)** & **bus_out_rsx (3:0)** & **Mux1 (3:0)** & **Mux2 (3:0)** & **bus_in_vrt (3:0)** & **Alu_OUT (3:0)**

La Figura A.4 muestra la simulación de la función scan de los DFT.

- Bypass_Scan: En este caso el DFT es similar en su funcionamiento al Scan pero solo actúa en el "Registro de Control". Debido a su complejidad e importancia el Registro de Control tiene un DFT dedicado. El registro de control tiene las instrucciones del código, 20 bits, provenientes de la ROM.

- Bypass: Este DFT permite la escritura del registro de control desde el exterior, dejando de lado la conexión con la ROM. La sincronización de este DFT es extremadamente necesaria

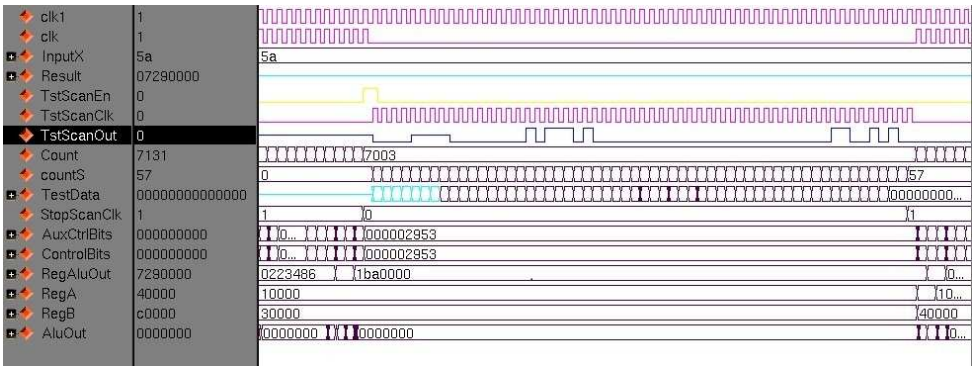

Figura A.4: Se demuestra la función "scan" en la simulación. Se observa que el reloj del chip esta detenido y la señal TstScanOut contiene la información que es capturada con la señal TstScanClk. El valor leído en TstScanOut es de derecha a izquierda y es el "53" que corresponde al valor del registro ControlBits.

para asegurar el funcionamiento correcto, por eso el reloj tiene que ser detenido. Los DFT Bypass_scan y Bypass fueron introducidos en el chip para una eventual falla de la ROM ya sea por un mal diseño ó una falla de manufactura. La idea era facilitar el procedimiento de validación ante un fallo de la ROM, se podía seguir verificando el chip introduciendo el código a ejecutar desde afuera y permitiendo evaluar otros posibles errores antes de un nuevo diseño. A continuación, la Fig. A.5 muestra una captura de la simulación donde es verificado este DFT.

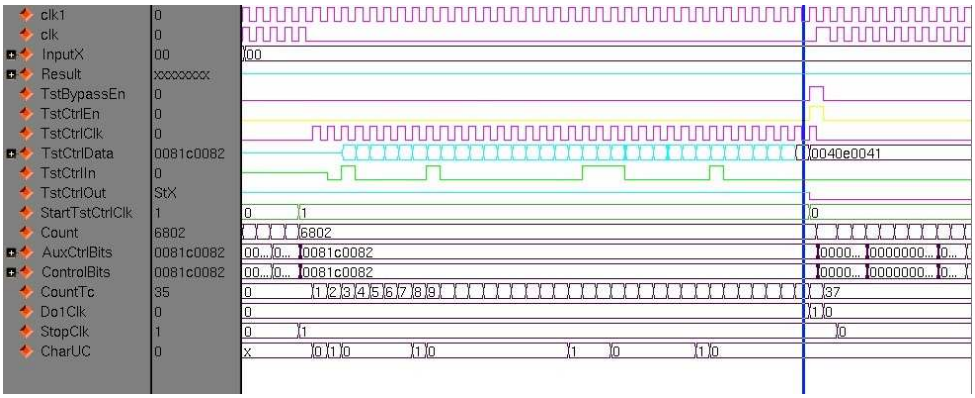

Figura A.5: Se observa que el tst_ctrl_data contiene, luego de ser cargado por el tst_ctrl_in, el mismo valor que el control_bits. (Se puede ver sobre el margen izquierdo). Claro que este valor se adquirió del control_bits para verificar que la simulación termina con el resultado correcto. Si este valor se elige diferente, se comprobó que la simulación no genero el resultado esperado.

A.5. Validación

En esta sección se detalla el proceso de validación y verificación del chip PWL. Se introduce el concepto de "power virus" para las medidas de consumo y de pass-fail test para la estimación

de las frecuencias máximas de operación. Se define la diferencia entre "massive" vs "focus" test en la parte de validación [58].

El primer paso en la parte de validación, es la verificación funcional por bloques. Esta verificación requiere que cada bloque sea analizado independientemente. Asegurar el correcto funcionamiento individual permite identificar los problemas cuando aparecen. Esta verificación se realizó utilizando un compilador para crear las distintas situaciones necesarias para la activación de cada bloque.

Los seis bloques principales incluidos en esta verificación son:

- DFT: Muchas de estas estructuras fueron verificadas durante el pre-silicio usando un analizador lógico. Debido a que este chip era el primer silicio que contaba con este paquete de validación, se dedicó una verificación exclusiva para estos DFT. Esto es importante porque serán las herramientas a utilizar para la verificación del chip.

- Máquinas de estado en la FPGA: Éstas fueron comprobadas utilizando un analizador lógico. Los protocolos y los tiempos fueron validados utilizando los resultados del analizador y las simulaciones correspondientes. Se utilizó las interfases de la FPGA de manera de simular la actuación del PWL y así se comprobó el funcionamiento correcto de las máquinas de estado.

- Memoria ROM: Fue validada utilizando los DFT para verificar la correcta escritura y lectura de la ROM. Este bloque era crítico para el funcionamiento del chip y no se había podido simular en las etapas de diseño ya que dependía de cómo la tecnología implementaba los latch utilizados para la creación de la memoria del PWL. La manera de verificar fue escribir distintos códigos y verificar ciertas direcciones que son observables con el mux de los DFT.

- Protocolo de los datos de entrada: Esta validación incluyó todos los protocolos asincrónicos entre el maestro y esclavo. Estos son: los datos de entrada y el resultado entregado por el PWL. La idea consistía en verificar la dirección de memoria que generaba el PWL ya que esta dependía directamente de los datos de entrada. Para esto se uso los DFT y el analizador lógico para garantizar la correcta comunicación.

- Clasificación (sorting): Debido a la fuerte dependencia con el valor n-dimensional se verificaron diferentes valores de entrada y utilizando los DFT de observabilidad se verificó el valor correcto de este bloque comparando con los resultados del simulador. La manera de proceder fue detectar las direcciones de memoria generadas por el PWL ya que éstas se hallan vinculadas a los datos de entrada y su ordenamiento.

- Acceso externo a memoria: Un código de programa específico se utilizó para leer y escribir a la memoria diferentes direcciones. Los datos fueron verificados usando el DFT y un analizador lógico.

- ALU: Además de los cálculos propios del modelo PWL, se verificaron diferentes operaciones matemáticas y los resultados se chequearon utilizando los DFT de observabilidad y el analizador lógico sacando los datos a través del mux.

Luego de la verificación por bloque, se realizó una verificación más integral abarcando el funcionamiento de todo el chip. Esta verificación se divide en dos procesos muy diferentes donde la diferencia reside en los datos utilizados. Los procesos son:

- Casos de simulación
- Casos Aleatorios

Donde se define como "caso", al conjunto de datos de entrada, en este ejemplo 6 variables, y se conoce el resultado del cálculo PWL. El resultado se produjo utilizando el Matlab donde se tenia una librería que implementa el cálculo de una estructura PWL. La diferencia esencial entre los casos de simulación y aleatorios reside en que los casos de simulación son pocos pero se conoce el estado de cada bloque en cada paso del reloj. Esto se hizo con el simulador y se logró sincronizar cada estado parcial del chip con el simulador. Esto se realizó utilizando los DFT de observabilidad y los de sincronización.

Por otro lado, los casos aleatorios se generan con el Matlab y lo único que se conoce es el resultado final, pero el número de casos aleatorios supera en ordenes de magnitud a los casos de simulación. En lo relativo a al validación, estos se conocen como "massive test" los aleatorios y "focus test" los de simulación. Es claro que puede existir algo intermedio entre ambos, pero eso requiere de un grado de automatización para verificar los resultados.

Para la validación de los casos aleatorios, se usó más de 50K puntos de R6 y se uso una función lineal de manera de reducir a 0 el error de aproximación entre la función y su aproximación PWL. El proceso consistía en generar de manera aleatoria los valores de entrada y luego comparar los resultados entre el Matlab y el chip. La Fig. A.6 muestra como se ve la comparación para algunos puntos.

Medición de Frecuencia máxima de operación

La idea de esta parte de la verificación fue determinar la frecuencia máxima de operación del chip PWL. La frecuencia de operación depende de la tensión de alimentación. Una limitante importante en el chip fue que la alimentación del core y del I/O era compartida. Esto impedía subir la tensión del core por arriba de los 3,3 Volt ya que se accionaban los clamping de la FPGA. A los efectos de superar esta limitación se procedió a definir un experimento para extrapolar la máxima frecuencia sin cambiar el voltaje de alimentación del chip por fuera del rango aceptable de los I/O de la FPGA.

El experimento consistió en ejecutar un cálculo PWL para distintas tensiones de alimentación del chip, manteniendo una frecuencia de operación. La tensión de alimentación se redujo hasta

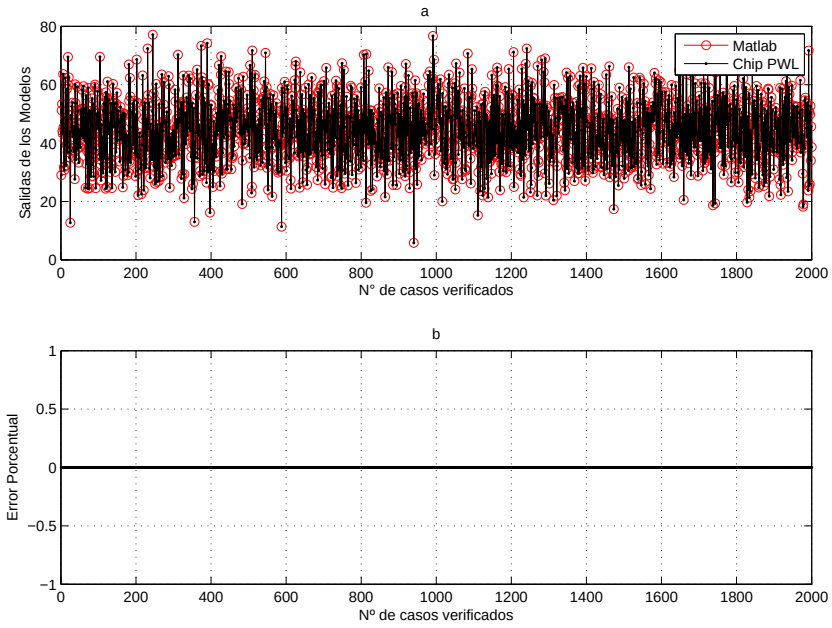

Figura A.6: a) Salida del PWL chip comparado con el PWL implementado en Matlab. b) Error absoluto obtenido de ambas salidas, siendo este igual a cero.

obtener un error funcional. Se entiende por error funcional a aquel resultado que difiere del correcto, pero en que el chip continúa operativo, es decir los protocolos entre el maestro y esclavo continúan funcionando. El fracaso del cálculo PWL es el punto que se marca en el gráfico de frecuencia vs. voltaje del chip. (La frecuencia del chip se cambió usando el bloque Digital Clock Manager (DCM) propietario de la FPGA). La Fig. A.7 presenta los puntos obtenidos para distintas tensiones de alimentación y frecuencia de operación. Donde los puntos medidos son los que se lograron en el laboratorio con el chip en funcionamiento y los puntos verificados son los que se comprobaron en el chip, luego de obtener la aproximación lineal, es decir, luego de verificar que la tensión y frecuencia estimada por la recta era adecuada.

Es conveniente clarificar un punto importante en la validación y es el concepto de "cubrimiento" (coverage) de los casos de validación. Es decir, cómo controlar todos los posible recorridos o combinaciones digitales para garantizar que el camino crítico que limita la frecuencia, fue testeado por el caso elegido. Este camino critico será el que limita la frecuencia de chip. Una manera práctica de solucionar el tema del cubrimiento es usar puntos aleatorios. Se supone que cuantos más casos se ejecuten, el cubrimiento tendría que mejorar. Cabe destacar que para el caso de este procesador específico, el camino critico estaba entorno a la ALU.

Para aplicar el concepto de cubrimiento, se generó en Matlab un conjunto de 5K puntos aleatorios y se realizó, nuevamente, el gráfico de tensión vs frecuencia. La Fig. A.8 muestra la diferencia obtenida cuando se cambia el cubrimiento. El cubrimiento débil son 9 casos y

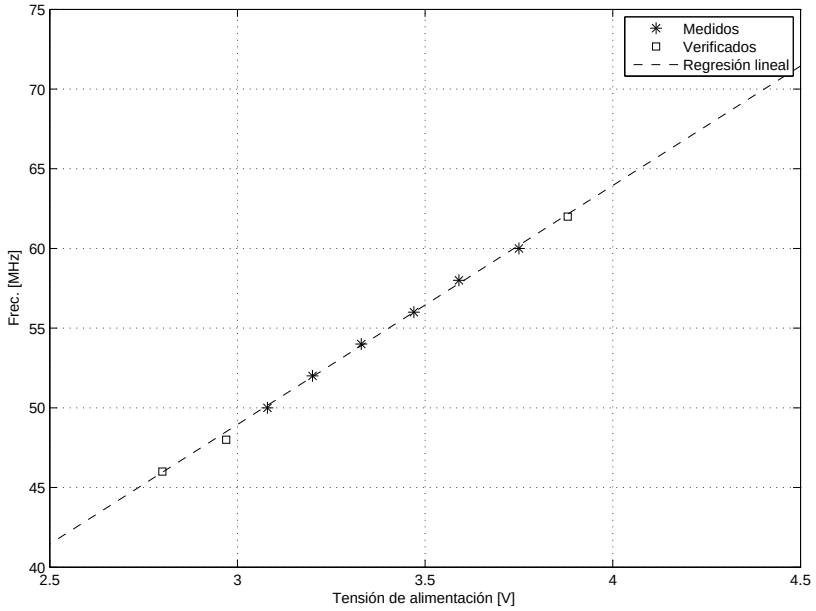

Figura A.7: Resultado del barrido de tensión y frecuencia para un conjunto, pequeño, de casos de prueba.

el cubrimiento fuerte son 5K casos por cada punto de tensión Vs Frecuencia. Como se puede apreciar, se tiene una diferencia en frecuencias debido al cubrimiento. De hecho, se encontró que el camino critico estaba en la parte de ordenamiento. Se comprobó que cambiando la parte fraccionaria de los datos de entrada el chip arrojaba un resultado erróneo. Si este chip tuviera una segunda fase de diseño, la parte del ordenamiento sería la que se modificaría para alcanzar mayores frecuencias.

Medición del Consumo

El consumo de potencias tiene tres componentes destacados: Consumo estático, árbol de reloj y consumo dinámico. El procedimiento para obtener estos componentes está resumido en el siguiente cuadro A.1: Donde la condición de "ejecutando" significa que el chip se encuentra

Cuadro A.1:
Definición de las 3 componentes del consumo del chip.

Componente	CLK	Reset	Ejecutando
Estático	off	on	off
Árbol de Reloj	on	on	off
Dinámico	on	off	on

procesando las instrucciones del código.

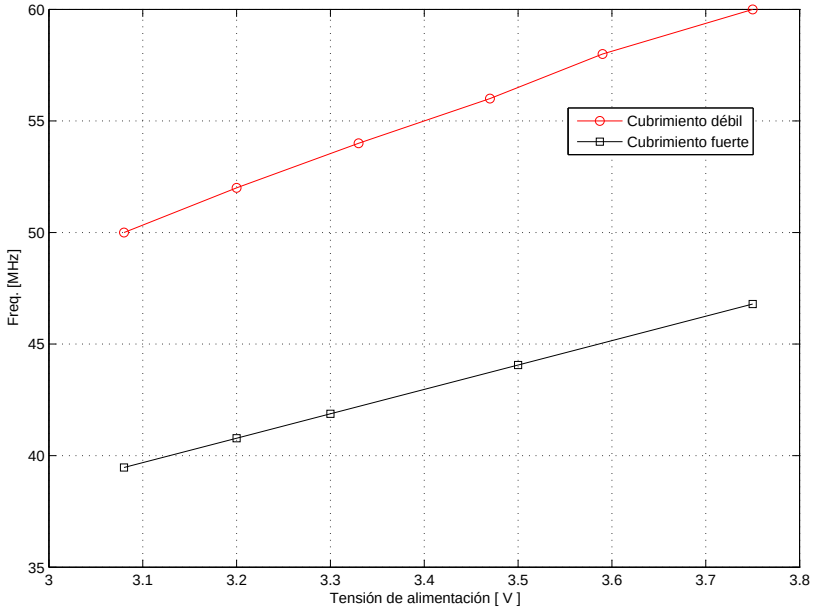

Figura A.8: Resultado del barrido de tensión y frecuencia para dos conjuntos de prueba con diferente cantidad de casos. La diferencia entre débil y fuerte se refiera al número de casos realizadas: el débil son 9 casos mientras que el fuerte son 5K.

El consumo de potencia se midió utilizando una resistencia en serie con la alimentación del chip. El valor de esta resistencia se calculó para que su voltaje esté dentro del rango [600mV, 1500mV]. El valor utilizado fue de 100Ω. Las mediciones se hicieron utilizando dos dispositivos: un osciloscopio digital (Agilent DSO3062A) y un multimetro Hewlett Packard (HP34401A). La idea fue la correlación de las mediciones con ambos elementos de medición.

Para tener una idea del consumo del chip durante la ejecución, se capturó con el osciloscopio una imagen, Fig. A.9, de manera de tener los consumos del chip para cada estado durante la ejecución. Se agregó la señal Req para mostrar la sincronización. Esta señal Req pertenece al protocolo asincrónico de entrada de datos y salida del cálculo PWL

Debido a que la ALU presenta el mayor consumo, se definió un "power virus". Este es un código desarrollado para que el PWL ejecute la ALU de manera recursiva realizando operaciones matemáticas. Además, se agregó la utilización de los pines de salida para considerar el consumo con este tipo de actividad.

Una vez obtenido el power virus, se procedió a cambiar la frecuencia de operación manteniendo la tensión de alimentación constante para obtener los distintos consumos, mostrado en el cuadro A.2. El consumo estático, de 160nW, se obtuvo de manera diferente ya que este corresponde al consumo propio del chip por perdidas en los transistores. Es interesante destacar que otra aplicación de estos Power virus es para calcular la disipación térmica del chip para el caso de un

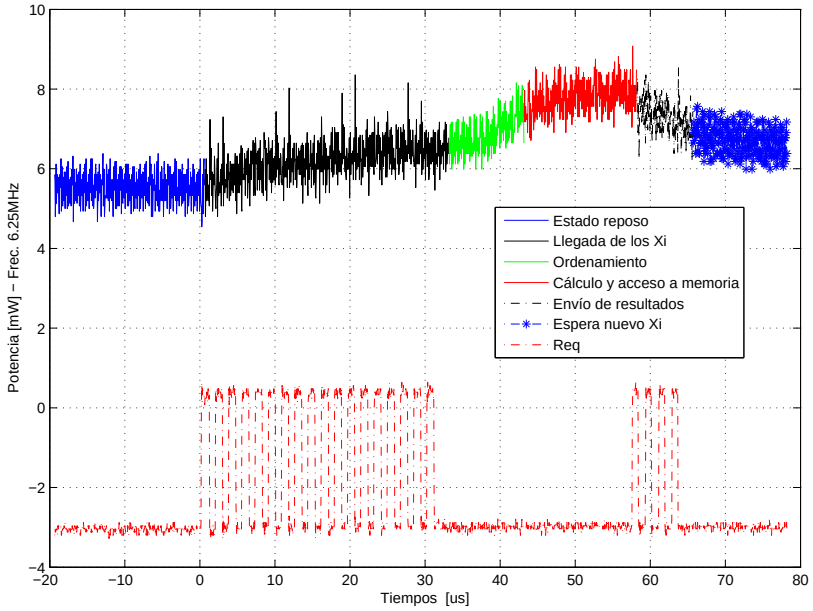

Figura A.9: Resultado del consumo para cada estado del chip durante la ejecución de un caso típico. Se puede ver que la ALU tiene el mayor consumo, como es de esperar en estos tipos de procesadores. La señal Req está fuera de escala y se usa solamente para visualización en el gráfico.

empaquetado cerrado.

Cuadro A.2:
Consumos de las componentes del chip para distintas frecuencias de operación para una tensión de alimentación de 3.3 Volt.

PWL	Potencia [mW]		
Frecuencia. [MHz]	Árbol de Reloj	P. Virus	P. Virus + I/O
25	22,70	34,91	49,68
12,5	11,42	17,60	24,87
6,25	5,69	8,75	12,40

Se puede observar el aumento lineal del consumo con la frecuencia.

A.6. Conclusiones

En este capítulo se describió el ASIC que implementa una estructura PWL. Para este chip se creó el entorno de funcionamiento que implicaba el desarrollo de un controlador digital Maestro-esclavo, la implementación de un controlador de memoria y de un adaptador de protocolos que contenía tres frecuencias de operación distinta.

Se introdujo un diseño digital especializado dentro del chip para la tareas de validación y

testeo. Se realizaron testeos funcionales y del chip completo. Se definieron distintos cubrimientos (focus y massive) y se verificaron las diferencias en los resultados. Se logró encontrar una limitante de frecuencia en la parte de ordenamiento del chip.

Se describieron técnicas para la medición del consumo del chip y frecuencias máximas de operación ante la limitación de los valores de tensiones de alimentación. Se definieron condiciones de éxito y fracaso para poder aplicar estas técnicas mencionadas. Se observaron las correlaciones de las potencias con ambos instrumentos de medición.

Apéndice B

B.1. Posible optimización de la arquitectura de identificación

A modo de ejemplo se hizo un esquema del bloque de identificación donde se aplica la ventaja del paralelismo para optimizar en velocidad. La Fig. B.1 muestra cómo serían los bloques de esta optimización. Esta es una implementación en paralelo mientras que la implementada finalmente en FPGA es una implementación tipo serie.

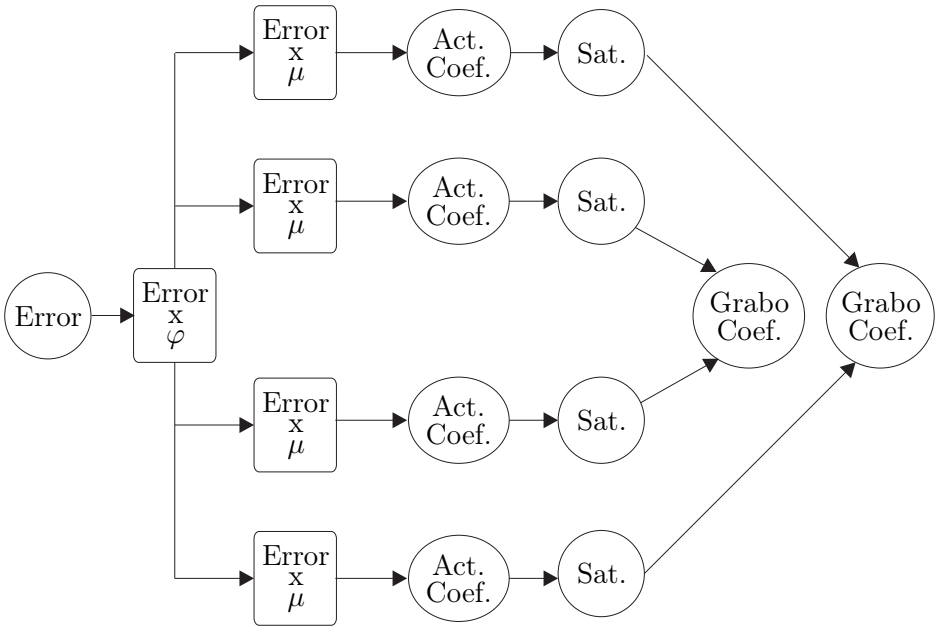

Figura B.1: Esquema de los estados para una identificación optimizada en velocidad. Se mejora la velocidad aumentando la cantidad de recursos. Se logra obtener la identificación en siete ciclos de reloj, suponiendo una memoria de doble puerto y para un caso de tres dimensiones.

De manera de poder comparar ambas implementaciones, se realizó un cuadro, B.1, que muestra las figuras de mérito para ambas.

Si bien se emplean más recursos en una estructura tipo paralelo, se logró realizar el proceso de identificación en siete ciclos de reloj. Esta optimización es relevante cuando el número de

Cuadro B.1:
Comparación de las figuras de mérito entre la implementación del algoritmo de identificación y una versión mejorada donde se quiere optimizar en velocidad para un caso de tres dimensiones.

Ítem	Implementado	Opt. en Velocidad
Lectura Memoria	4	0
Escritura Memoria	4	2
Ciclos de Reloj	22	7
$N°$ Multiplicadores	1	4
$N°$ Sumadores	1	4
$N°$ Restadores	1	1
$N°$ Comparadores	2	8

datos en un simplice es elevado en el caso de una identificación instantánea.

B.2. Posible estructura de control

En esta sección se describe un posible esquema de control utilizando una estructura NOE y cómo sería el procedimiento de identificación "On Lineť".

La Figura B.2, muestra un esquema de control donde se duplica el HW existente para ir ajustando en linea los parámetros del modelo NOE, ante los cambios del sistema, bajo las acciones de control mientras que, la otra estructura NOE, cierra el lazo de control. El cambio entre los dos estructuras NOE se podría implementar según el error entre el modelo NOE y el sistema. Dado un error especifico, se alterna la estructura NOE que cierra el lazo de control

Otra forma de lograr lo mismo, sería tener un HW suficientemente rápido para poder hacer la identificación y el control del sistema en la misma estructura NOE.

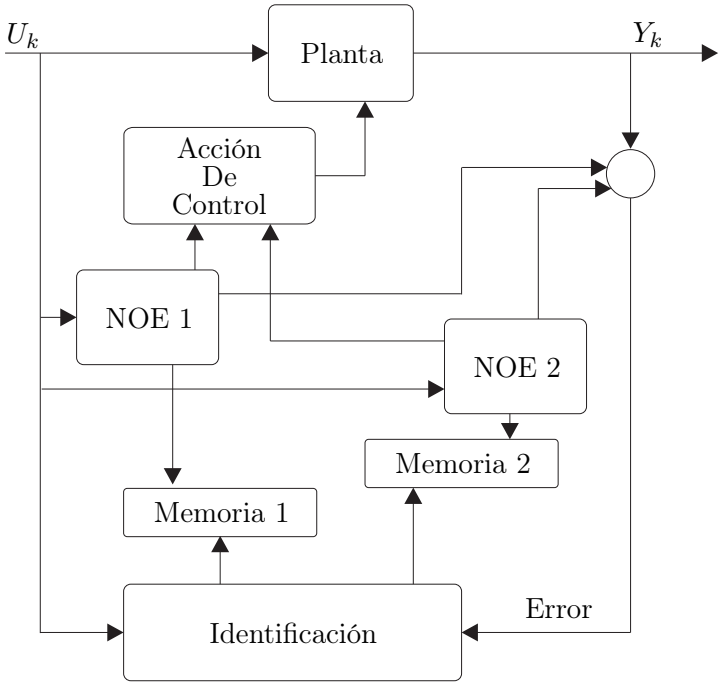

Figura B.2: Posible esquema de control. Se alterna la estructura NOE que interviene en el control del sistema mientras se ajusta el nuevo modelo del sistema en la otra estructura NOE.

Apéndice C

En este apéndice se describen tres elementos que se analizaron durante de desarrollo del trabajo presentado. Estos elementos son: las memorias disponibles en la placa de desarrollo FPGA, El Adaptador de Protocolos, AdP, entre el chip PWL y el controlador provisto por la FPGA y la efectividad de la implementación de una una memoria tipo "cache". Además se presentan algunas medidas de desempeño del controlador DDR y de las memorias integradas propias del dispositivo FPGA.

C.1. Comentarios sobre la memoria

La mayoría de las FPGA contienen bloques, denominados generalmente Intellectual Property (IP), disponibles para el usuario final. Estos IP están optimizados para la tecnología de la FPGA. Demás esta decir, que si el desarrollo es transportado a otro dispositivo FPGA, estos IP pueden variar o, inclusive, no existir.

Para el caso de la Virtex5, la implementación de la memoria DDR fue realizada con la herramienta provista por el Memory Inteface Generator (MIG). Este provee un controlador de memoria para la DDR y la interfase para poder utilizarlo en el diseño, esta interfase es mostrada en la Fig. C.1.

La Figura C.2 muestra las señales que provee el controlador de memoria para su vinculación con el mundo exterior. Estas señales son las que se conectan con el AdP para adecuar el protocolo que maneja el chip PWL y el controlador propiamente dicho. Es claro que el controlador maestro, también, tiene acceso a memoria, pero lo hace directamente sin utilizar el AdP.

Una característica importante de las memoria comerciales DDR es su manejo automático del "burst", esto significa que para una dada dirección de memoria, su controlador provee los datos de las 8 direcciones siguientes en orden creciente a la dirección pedida. Este burst es resultado directo de la forma de programación continua para lo cual las instrucciones del programa y datos estarán en forma continua contenidos en la memoria. Los datos obtenidos durante el burst son almacenados en una memoria interna, denomina "Cache".

Para el caso de la estructura PWL, en las aplicaciones del ADC y del modelo NOE esta característica no se puede aprovechar por la forma en la que las direcciones de memoria son

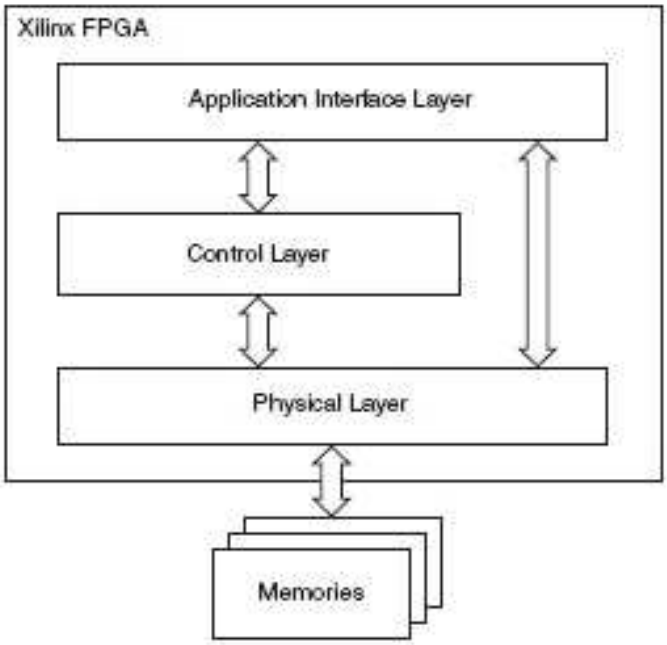

Figura C.1: Capas de la interfase del controlador de memoria provisto por la herramienta de sintetización. Figura adquirida del manual de memorias de Xilinx

generados, por lo tanto, la opción de Cache no es efectiva como se verá más adelante.

Para poder medir el desempeño del dispositivo de memoria y el controlador de memoria en conjunto, se diseño un caso para lo cual se media la mayor transferencia de datos del sistema. Este consistía en generar direcciones de memoria de distinta formas y medir la velocidad de transferencia de los datos enviados por el controlador. El primer dato importante fue estimar el tiempo de respuesta (Latency) que tenia el conjunto: controlador y memoria. Este fue de 26 ciclos de reloj y este número fue constante para todas las pruebas realizadas con distinto tipos de direcciones de memoria. La Fig. C.3 muestra las velocidades obtenidas en el controlador y el dispositivo físico de memoria. Las direcciones secuenciales son aquellas donde se utiliza el burst y se aprovecha el ancho de banda de la memoria y las No-secuenciales son las direcciones que no guardan ninguna relación entre ellas. Es decir, solamente se aprovecha un byte de los 64 entregados por el controlador de memoria. En resumen, se vio que para direcciones No-secuenciales el orden de transferencia está en el orden del 1,5M bits.

Otro tipo de memoria que se analizó fueron las memorias integradas, Static Random Access Memory (SRAM), en la FPGA. Estas memorias son muchos mas rápidas pero de mucho menor tamaño. El tamaño es del orden de K Bytes en comparación con la DDR que son de Mega Bytes. Otro factor relevante de estas memorias integradas es su tiempo de respuesta, que es de un ciclo de reloj. Siendo este el más rápido que se puede obtener.

La manera de poder evaluar la velocidad máxima obtenible en estas memorias integradas en

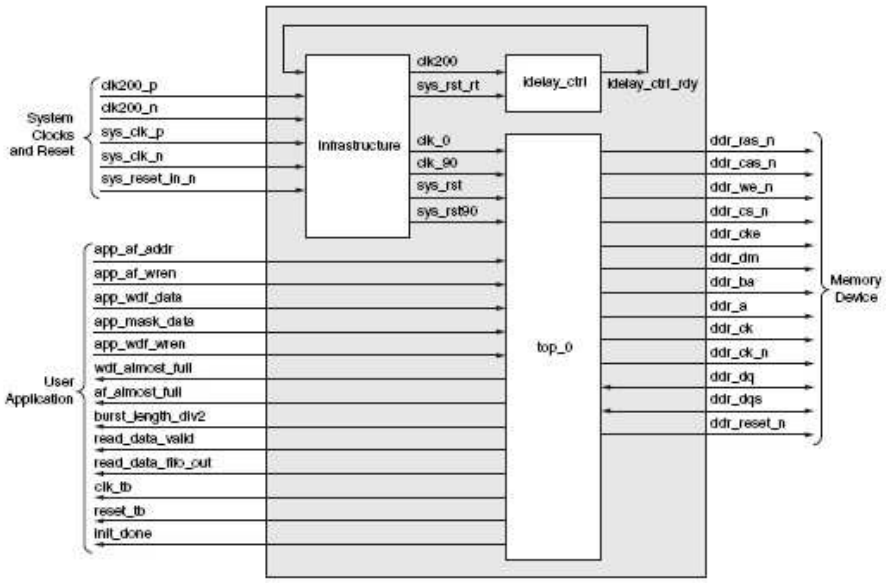

Figura C.2: Señales provistas por el controlador de memoria hacia la interfase con el mundo exterior y hacia el dispositivo de memoria. Provisto por el manual de Xilinx.

la FPGA es hacer la implementación de sistema que escribe una cierta cantidad de direcciones con un dato especifico y luego se hace un ciclo de lectura de estas mismas direcciones, verificando que el dato leído coincida con el dato esperado.

Estos ciclos de escritura y lectura se hacen repetidamente cambiando la frecuencia de operación para cada evento de escritura y lectura. De esta manera cuando obtenemos un error en la lectura consideramos que estamos en una frecuencia por arriba de la frecuencia de operación. El número que se obtuvo fue del orden de 250MHz de frecuencia de operación para la Virtex5 que se analizó. La forma de obtener esta memoria en la FPGA es usando la función "coregen" incluida en el entorno del sintetizador propio de la FPGA. Demás esta decir que el protocolo de manejo de estas memorias es muy simple y fácil de codificar en VHDL, haciendo que la complejidad de AdP sea reducida en comparación a cuando se usa la memoria DDR.

C.2. Adaptador de Protocolos (AdP)

La implementación del AdP para adaptar el protocolo del chip PWL al controlador de memoria, se hizo considerando la forma de presentación de las direcciones provistas por el chip PWL. Las direcciones consisten en un registro de 16 bits con dos fases de direcciones: alta y baja, marcadas con una señal de control. La dirección final contiene 24 bits que fueron los 16 bits bajos y 8 bits altos. Por otro lado, el controlador de memoria del MIG opera con direcciones que tiene los dos bits menos significativos en 0 de manera de alinear las direcciones con el proceso automático del burst. La función de este AdP es transformar las direcciones y obtener el dato de la dirección

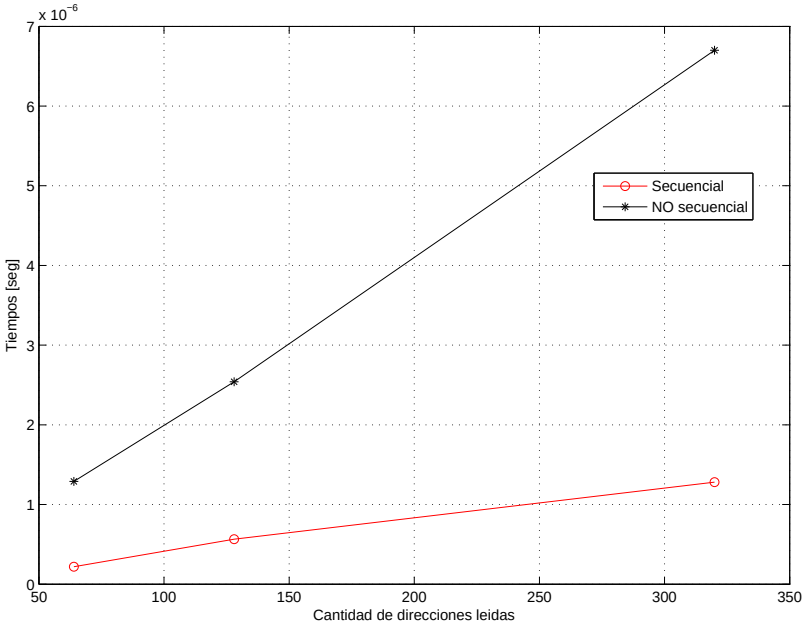

Figura C.3: Medidas de tiempos para leer las direcciones de memoria para direcciones secuenciales y no secuenciales. La diferencia es cuando se hace efectivo el uso del burst propio de la memoria.

pedida entre todos los datos que la DDR provee. Es decir, dada una dirección de memoria del chip, el AdP registra cual era la dirección original y modifica la dirección de manera de cumplir con las restricciones de los últimos dos bits. Una vez recibido todos los datos del burst, el AdP solamente toma el dato pedido, descartando el resto.

C.3. Uso del cache

Para poder mejorar el rendimiento del sistema controlador y memoria, se hizo un análisis del uso de un cache interno con la idea de mejorar los tiempos de acceso de la memoria externa.

Para esto, se obtuvieron las direcciones de memoria generadas por el PWL para una implementación del ADC y se estudió la correlación de dichas direcciones de memoria. Ademàs se utilizó un sistema de cache simulado en Matlab, de manera de poder estudiar su rendimiento.

La forma con que el chip PWL presenta las direcciones de memoria es la siguiente: la parte entera del dato actual de entrada y, concatenado, las partes enteras de los datos pasados. La cantidad de muestras hacia atrás dependerá del orden o dimensión de la estructura PWL. Esto se ejemplifica en la Ec. C.4 donde se utilizó un R4 con tres muestras hacia atrás para el instante k.

$$
\begin{aligned}
direc_{(k)} &= [x_k, x_{k-1}, x_{k-2}, x_{k-3}] \\
direc_{(k+1)} &= [x_{k+1}, x_k, x_{k-1}, x_{k-2}] \\
direc_{(k+2)} &= [x_{k+2}, x_{k+1}, x_k, x_{k-1}]
\end{aligned}
\tag{C.1}
$$

Observando un ejemplo real del ADC las direcciones, para un instante en particular, serían las contenidas en la Fig. C.4:

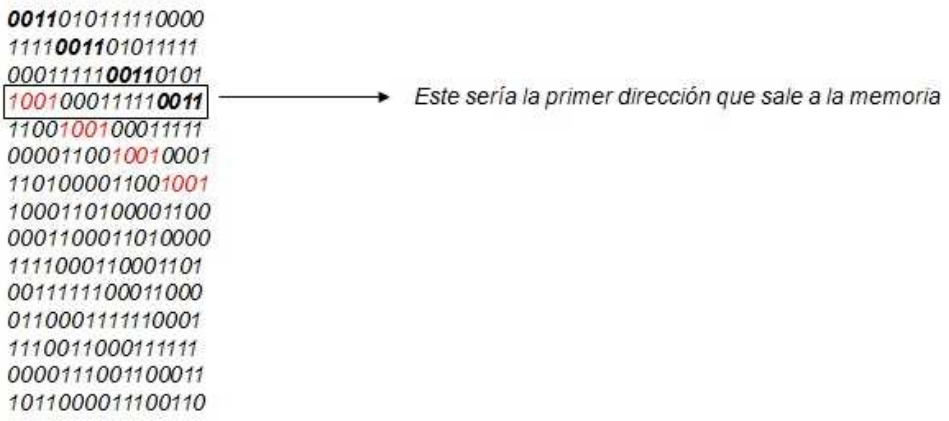

Figura C.4: Se observa como las direcciones generadas por el PWL no guardan ninguna relación entre ellas, por lo tanto el burst queda sin efecto

Para simular el efecto de un cache, se realizó un programa, en Matlab, para estimar el tamaño y su utilización para un conjunto de direcciones reales.

El cache esta compuesto por un número variable de bloques y un número variable de la cantidad de direcciones que quedan dentro de un burst, para el caso de una DDR comercial este número es 4 u 8, pero para el caso de estudio, este número se puede cambiar.

Luego para cada dirección leída, se modifica el bloque correspondiente y se modifica el valor de las variables "hit" y "miss". La variable hit en 1 representa el momento en que la dirección requerida se encuentra dentro del cache porque ésta fue adquirida por un evento anterior. La variable miss en 1, indica que la dirección no se encuentra y esto hace que se tenga que salir a la memoria exterior para buscar el dato. En el caso del "miss", el cache no cumplió ninguna función. La Fig. C.5 muestra la cantidad de "hit" y "miss" producidos para el caso de una estructura PWL aplicada a la corrección de un ADC.

Se pude ver que la tasa de hits es muy pequeña para tamaños considerados de cache. Esto enfatiza la desestimación de cache para nuestra aplicación.

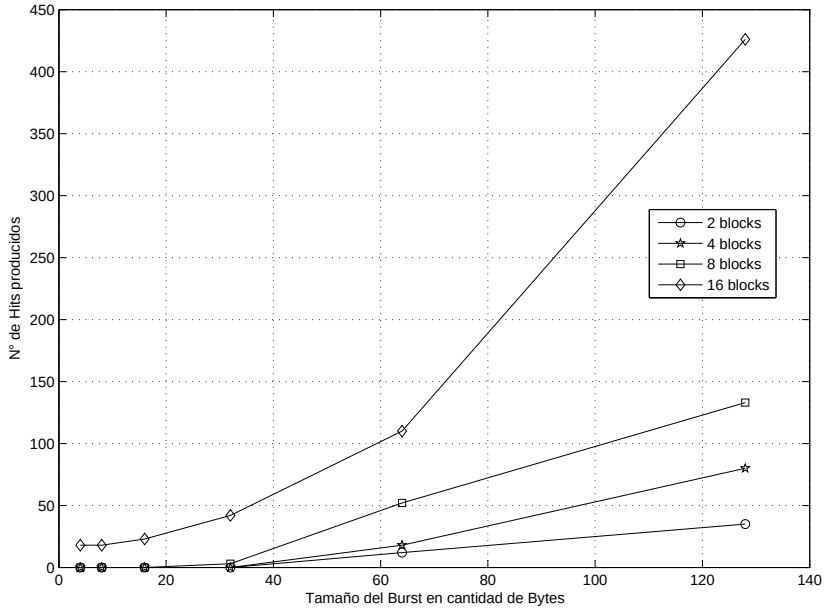

Figura C.5: Se representa la cantidad de "Hits" para un conjunto de direcciones reales producidas por una estructura PWL en la aplicación de corrección un dispositivo ADC. La cantidad de direcciones leídas fue de 1199. Se ve que la implementación de un cache sería poco utilizada por la baja cantidad de hits.

Bibliografía

[1] J. de Boeij, E. Lomonova, and A. Vandenput. Look-up table based real-time commutation of 6-dof planar actuators. *in Proc. IEEE Int. Conf. Control Appl*, pages 1118–1123, 2007.

[2] J. Detrey and F. de Dinechin. Table-based polynomials for fast hardware function evaluation. *in Proc. 16th IEEE Int. Conf. Appl.-Specific Syst., Archit. Processor (ASAP 2005)*, pages 328–333, 2005.

[3] A. Strollo, D. D. Caro, and N. Petra. A630mhz, 75mwdirect digital frequency synthesizer using enhanced rom compression technique. *IEEE J. Solid-State Circuits*, 42(2):350–360, 2007.

[4] T. Mestl, C. Lemay, and L. Glass. Chaos in high dimensional neural and gene networks. *Physica D*, 98(1):33–52, Nov. 1996.

[5] J. Eckmann and E.Moses. Curvature of co-links uncovers hidden thematic layers in the world-wide web. *P. Nat. Acad. Sci. USA*, 99(9):5825–5829, Apr. 2002.

[6] W. B. Arthur. Complexity and the economy. *Science*, 284(5411):107–109, Apr. 1999.

[7] B. Carreras, V. Lynch, I. Dobson, and D. Newman. Critical points and transitions in an electric power transmission model for cascading failure blackouts. *Chaos*, 12(4):985–994, Dic. 2002.

[8] S. Kang and L. Chua. A global representation of multidimensional piecewise-linear functions with linear partitions. *IEEE Trans. Circuits Syst.*, 25(11):938–940, 1978.

[9] M.-J. Chien and E. Kuh. Solving nonlinear resistive networks using piecewise-linear analysis and simplicial subdivision. *IEEE Trans. Circuits Syst.*, 24(6):305–317, 1977.

[10] L. Chua and R. Ying. Canonical piecewise-linear analysis. *IEEE Trans. Circuits Syst.*, 30(3):125–140, 1983.

[11] J.-A. Pineiro, S. Oberman, J.-M.Muller, and J. Bruguera. High-speed function approximation using a minimax quadratic interpolator. *IEEE Trans. Comput.*, 54(3):304–318, 2005.

[12] Pablo S Mandolesi, Pedro Julian, and Andreas G Andreou. A scalable and programmable simplicial CNN digital pixel processor architecture. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 51(5):988–996, May 2004.

[13] M Di Federico, P Julian, P S Mandolesi, and A G Andreou. PWL cores for nonlinear array processing. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 3312–3316. IEEE, 2010.

[14] M Di Federico, P Julian, and P S Mandolesi. SCDVP: A Simplicial CNN Digital Visual Processor. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(7):1962–1969, 2014.

[15] M Di Federico, P Julian, A G Andreou, and P S Mandolesi. Fully functional fine-grain vertically integrated 3D focal plane neuromorphic processor. In *SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S), 2014 IEEE*, pages 1–2. IEEE, 2014.

[16] J. Rodríguez, O. Lifschitz, V. Jiménez-Fernández, P. Julián, and O. Agamennoni. Application-specific processor for piecewise linear functions computation. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 58(5):971–981, may 2011.

[17] J. Yuan, N. Farhat, and J. V. der Spiegel. Background calibration with piecewise linearized error model for cmos pipeline a/d converter. *IEEE Trans. Circuits Syst. I, Reg. Papers*, 55(1):311–321, 2008.

[18] R. Hiorns and M. Sandler. Power digital to analogue conversion using pulse width modulation and digital signal processing. *IEE Proc. Circuits, Dev. Syst.*, 140(5):329–338, 1993.

[19] A. Prodic and D. Maksimovic. Design of a digital pid regulator based on look-up tables for control of high-frequency dc-dc converters. *in Proc. IEEE Workshop Comput. Power Electron.*, pages 18–22, 2002.

[20] M. Sonnaillon, G. Bisheimer, C. D. Angelo, J. Solsona, and G. Garcia. Mechanical-sensorless induction motor drive based only on dc-link measurements. *IEE Proc. Elect. Power Appl.*, 153(6):815–822, 2006.

[21] O. Hammi, F. Ghannouchi, S. Boumaiza, and B. Vassilakis. A databased nested lut model for rf power amplifiers exhibiting memory effects. *IEEE Microw. Wireless Compon. Lett.*, 17(10):712–714, 2007.

[22] P. Chancharoensook and M. Rahman. Dynamic modeling of a fourphase 8/6 switched reluctance motor using current and torque look-up tables. *in Proc. IEEE 28th Annu. Conf. Ind. Electron. Soc. (IECON02)*, 1:491–496, 2002.

[23] M. Duran, S. Toral, F. Barrero, and E. Levi. Real-time implementation of multi-dimensional five-phase space vector pwm using look-up table techniques. *in Proc. 33rd Annu. Conf. IEEE Ind. Electron. Soc. (IECON 2007)*, pages 1518–1523, 2007.

[24] Z. Muhammad, Z. Yusoff, M. Rahiman, and M. Taib. Steam temperature control for steam distillation pot using model predictive control. In *Signal Processing and its Applications (CSPA), 2012 IEEE 8th International Colloquium on*, pages 474 –479, march 2012.

[25] G. Suarez, O. Ortiz, P. Aballay, and N. Aros. Adaptive neural model predictive control for the grape juice concentration process. In *Industrial Technology (ICIT), 2010 IEEE International Conference on*, pages 57 –63, march 2010.

[26] A. Imura, T. Takahashi, M. Fujitsuna, T. Zanma, and M. Ishida. Instantaneous-current control of pmsm using mpc: Frequency analysis based on sinusoidal correlation. In *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, pages 3551 –3556, nov. 2011.

[27] M. Rivera, V. Yaramasu, J. Rodriguez, and Wu Bin. Model predictive current control of two-level four-leg inverters; part ii: Experimental implementation and validation. *Power Electronics, IEEE Transactions on*, 28(7):3469 –3478, july 2013.

[28] F. Sajjad and A. Francis. Fast model predictive control and its application to energy management of hybrid electric vehicles. In *Advanced Model Predictive Control*, July 2011.

[29] O. Hammi, S. Boumaiza, M. Jaidane-Saidane, and F. Ghannouchi. Digital subband filtering predistorter architecture for wireless transmitters. *IEEE Trans. Microw. Theory Tech.*, 53(5):1643–1652, 2005.

[30] L.-C. Chang and J. V. Krogmeier. A look-up table with amplitude scaling technique for amplifier linearization. *in Proc. 10th IEEE Singapore Int. Conf. Commun. Syst. (ICCS 2006)*, pages 1–5, 2006.

[31] M. Bruno, S. Werner J. Cousseau, J. Figueroa, M. Cheong, and R Wichman. An efficient cs-cpwl based predistorter. *Radioengineering*, 18(2):170–177, 2009.

[32] W.-J. Kim, K.-J. Cho, S. Stapleton, and J.-H. Kim. Piecewise pre-equalized linearization of the wireless transmitter with a doherty amplifier. *IEEE Trans. Microw. Theory Tech.*, 54(9):3469–3478, 2006.

[33] M. Zakhama and D. Massicotte. A systolic architecture for channel equalization based on a piecewise linear fuzzy logic algorithm. *in Proc. IEEE Can. Conf. Elect. Comput. Eng.*, 2:1098–1101, 1999.

[34] M. Vidal and D. Massicotte. A vlsi parallel architecture of a piecewise linear neural network for nonlinear channel equalization. *in Proc. 16th IEEE Conf. Instrum. Meas. Technol. (IMTC/99)*, 3:1629–1634, 1999.

[35] R. Rovatti, A. Ferrari, M. Borgatti, A. Kandel, and G. Langholz. Automatic implementation of piecewise-linear fuzzy systems addressing memory-performance trade-off. *in Fuzzy Hardware: Architectures and Applications. Norwell, MA: Kluwer Academic*, pages 159–179, 1998.

[36] P. Echevarria, M. Martínez, J. Echanobe, I. del Campo, and J. Tarela. Digital hardware implementation of high dimensional fuzzy systems. *in Applications of Fuzzy Sets Theory, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer*, pages 245–252, 2007.

[37] M. Storace and M. Parodi. Towards analog implementations of pwl two-dimensional nonlinear functions. *Int. J. Circuit Theory and Applications*, 33(2):147–160, Mar.-Apr. 2005.

[38] M. Storace and T. Poggi. Şdigital architectures realizing piecewiselinear multi-variate functions: Two fpga implementations. *Int. J. Circuit Theory and Applications*, 39(1):1–15, 2011.

[39] F. Comaschi, B. A. G. Genuit, A. Oliveri, W. P. Maurice, H. Heemels, and M. Storace. Fpga implementations of piecewise affine functions based on multi-resolution hyperrectangular partitions. *IEEE Transaction On Circuits And Systems*, 59(12):2920–2933, Dec. 2012.

[40] P. Julian, A. Desages, and B. D'Amico. Orthonormal high-level canonical pwl functions with applications to model reduction. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 47(5):702–712, may 2000.

[41] P. Julian, A. Desages, and O. Agamennoni. High-level canonical piecewise linear representation using a simplicial partition. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 46(4):463–480, apr 1999.

[42] E. Sontag. Nonlinear regulation: The piecewise linear approach. *Automatic Control, IEEE Transactions on*, 26(2):346–358, apr 1981.

[43] S. Billings, H. Jamaluddin, and S. Chen. Properties of neural networks with applications to modelling non-linear dynamical systems. *Internat. J. Control*, 55(1):319–346, apr 1992.

[44] C. Fantuzzi, S. Simani, S. Beghelli, and R. Rovatti. Identification of picewise affine models in noisy environment. *Internat. J. Control*, 75(18):1472–1485, may 2002.

[45] S. Chen and S. Billings. Representation of non-linear systems: The narmax model. *Internat. J. Control*, 49(3):1013–1032, 1989.

[46] R. Haber and H. Unbehauen. Structure identification of nonlinear dynamic systems-a survey on input/output approaches. *Automatica*, 26(4):651–677, 1990.

[47] T. Johansen and B. Foss. Constructing narmax models using armax models. *Internat. J. Control*, 58(5):1125–1153, 1993.

[48] J. Sjioberg, Q. Zhang, L. Ljung, A. Beneviste, B. Delyon, P. Glorennec, H. Hjalmarsson, and A. Juditsky. Nonlinear black-box modeling in system identification: A unified overview. *Automatica*, 35(12):1691–1724, 1995.

[49] L. Castro, O. Agamennoni, and M. Álvarez. From linear to nonlinear identification: One step at a time. *Mathematical and Computer Modelling*, 51(5Ű6):473 – 486, 2010.

[50] R. V. de Plassche. Cmos integrated analog-to-digital and digital-to-analog converters. *Dordrecht, The Netherlands:Kluwer Academic Publishers*, 2003.

[51] E. Balestrieri, P. Daponte, and S. Rapuano. A state of the art on adc error compensation methods. *IEEE Transactions on Instrumentation and Measurement*, 54(4):1388–1394, 2005.

[52] H. Lundin. *Characterization and Correction of Analog-to-digital Converters*. PhD thesis, Stockholm, Sweden: Doctoral Thesis in Signal Processing, 2005.

[53] Y. Oh and B. Nurmann. System embedded adc calibration for ofdm receivers. *IEEE Transactions on Circuits and Systems*, 53(8):1693–1703, 2006.

[54] L. Vito, H. Lundin, and S. Rapuano. Bayesian calibration of a lookup table for adc error correction. *IEEE Transactions on Instrumentation and Measurement*, 56(3):873–878, 2007.

[55] F. Irons, D. Hummels, and S. Kennedy. Improved compensation for analog-to-digital converters. *IEEE Transactions on Circuits and Systems*, 38(8):958–961, 1991.

[56] O. Agamennoni O. Lifschitz, P. Julián. Accuracy analysis for on-chip digital pwl realization. In *Reunión de trabajo en Procesamiento de la Información y Control*, 2011.

[57] P. Julian. A high-level canonical piecewise linear representation: Theory and applications. In *UMI, Bell & Howell Inf. Learn.*, page 182, 1999.

[58] O. Lifschitz, J. Rodriguez, P. Julian, and O. Agamennoni. Post-silicon validation procedure for a pwl asic microprocessor architecture. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 9(4):492–497, july 2011.

Glosario

ADC Analog to Digtal Converter. 3, 37–45, 49–60, 76–79, 84, 87, 95, 96, 115, 118–120

AdP Adaptador de Protocolos. 99, 115, 117, 118

ALU Arithmetic Logic Unit. 97, 105, 106, 108, 109

ARMAX Autoregressive Moving Average with eXogenous inputs. 5, 7

ASIC Application-Specific Integrated Circuit. 2, 3, 9, 23–27, 29, 97, 109

DCM Digital Clock Manager. 106

DDR Double Data Rate. 62, 99, 100, 115–119

DFT Design For Testability. 97, 98, 100–105

DRAM Dynamic Random-Access Memory. 2

DSP Digital Signal Processors. 2

ECM Error Cuadrático Medio. 20, 21, 25, 29–31, 33, 56

EDA Electronic Design Automation. 9, 97

ENOB Effective Number of Bits. 51–57, 59, 60, 96

FIFO First In First Out. 42, 90

FIR Finite Impulse Response. 3, 8, 9, 38, 39, 46–48, 52, 56, 58, 60, 75, 79, 80, 87, 88, 91, 95, 96

FPGA Es un dispositivo electrónico (del ingles Field Programmable Gate Array), que contiene bloques de lógica cuya interconexión puede ser configurada mediante un lenguaje de descripción de hardware. La lógica utilizada puede incluir desde funciones sencillas hasta complejos sistemas en chip. Las FPGAs se utilizan en aplicaciones similares a los ASICs sin embargo son más lentas, tienen un mayor consumo de potencia y no pueden abarcar sistemas tan complejos. A pesar de esto, tienen las ventajas de ser re-programables y sus

costos de desarrollo son mucho menores. 3, 13, 15, 17, 23, 29, 32–35, 76, 87, 90, 92, 95–97, 99, 104–106, 111, 115–117

GISEE Grupo de Investigación en Sistemas Electrónicos y Electromecatrónicos. 2, 9, 21

GMICS Grupo de Modelado, Identificación y Control de Sistemas. 2

GPU Graphic Processor Unit. 2

HW Hardware. 2, 21, 80, 86, 95, 96, 98, 99, 112

INL Integral Nonlinearity. 42

IP Intellectual Property. 115

ISA Instruction Set Architecture. 97

LSB Low Significant Bit. 42

LUT Look-Up Table. 1, 2

MIG Memory Inteface Generator. 115, 117

MPC Model Predictive Control. 96

MSPS Mega-Samples Per Second. 42

NARMAX Nonlinear Autoregressive Moving Average with eXogenous inputs. 7

NARX Auto-Regressive with eXogenous input. 5, 7

NARX Nonlinear Auto-Regressive with eXogenous input. 8

NFIR Nonlinear Finite Impulse Response. 9

NOE No Linear Output Error. 3, 5, 7–9, 13, 14, 18, 21–33, 35, 41, 75, 79, 80, 86–88, 90, 92, 95, 96, 112, 113, 115

OE Output Error. 5, 7, 8

PC Personal Computer. 23, 24, 30, 42, 56, 59, 88, 90, 99–101

PCB Printed Circuit Board. 99

PicoBlaze El microcontrolador Picoblaze es un Soft Core de 8 bits, diseñado para ser empotrado en dispositivos FPGAs.. 90

PID Proporcional Integrativo Derivativo. 1

PWL Piecewise Linear. 1–3, 6, 9, 10, 13, 14, 17, 19, 20, 22, 23, 31, 35, 37–41, 44, 48, 52–54, 56–58, 60–62, 65–70, 73–80, 82, 83, 87, 88, 90–93, 95–100, 103–106, 108, 109, 115, 117–120

ROM Read Only Memory. 100, 102–104

RS232 Es un estándar electrónico (del ingles Recommended Standard 232), utilizado para la transmisión serie de datos binarios y señales de control. Fue diseñado para conectar un DTE (equipo terminal de datos) y un DCE (equipo de comunicación de datos). Es comúnmente utilizado en los puertos serie de las computadoras modernas. 88–90, 99, 100

SDRAM Synchronous Dynamic Random-Access Memory. 62

SINAD Signal to Noise and Distortion ratio. 40, 44, 45, 49–54, 60

SNR Signal to Noise Ratio. 49, 50

SRAM Static Random Access Memory. 116

SW Software. 42, 44, 98–100

VHDL Es un lenguaje de descripción de hardware (HDL, del ingles Hardware Description Language), usado para modelar sistemas electrónicos. Soporta el diseño, prueba e implementación de circuitos analógicos, digitales y de señal mixta a diferentes niveles de abstracción. 87, 100, 117

VNA Vector Network Analizer. 42