Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE ADQUISICIÓN DE DATOS PARA UN DETECTOR GRB EN GUATEMALA

Luis Guillermo García Ordóñez
Asesorado por el Ing. Walter Giovanni Álvarez Marroquín

Guatemala, agosto de 2015

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE ADQUISICIÓN DE DATOS PARA UN DETECTOR GRB EN GUATEMALA

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

LUIS GUILLERMO GARCÍA ORDÓÑEZ

ASESORADO POR EL ING. WALTER GIOVANNI ÁLVAREZ MARROQUÍN
AL CONFERÍRSE EL TÍTULO DE

INGENIERO EN ELECTRÓNICA

GUATEMALA, AGOSTO DE 2015

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Pedro Antonio Aguilar Polanco
VOCAL I	Ing. Angel Roberto Sic García
VOCAL II	Ing. Pablo Christian de León Rodríguez
VOCAL III	Inga. Elvia Miriam Ruballos Samayoa
VOCAL IV	Br. Narda Lucía Pacay Barrientos
VOCAL V	Br. Walter Rafael Véliz Muñoz
SECRETARIA	Inga. Lesbia Magalí Herrera López

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

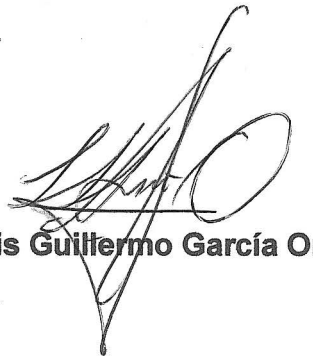
DECANO	Ing. Murphy Olympo Paiz Recinos
EXAMINADOR	Ing. Carlos Eduardo Guzmán Salazar
EXAMINADORA	Inga. María Magdalena Puente Romero
EXAMINADOR	Ing. Romeo Neftalí López Orozco
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE ADQUISICIÓN DE DATOS PARA UN DETECTOR GRB EN GUATEMALA

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha 3 de mayo de 2013.



Luis Guillermo García Ordóñez

Guatemala, 26 de enero de 2015

Ingeniero
Carlos Eduardo Guzmán Salazar
Coordinador de Área de Electrónica
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería
Universidad de San Carlos de Guatemala

Señor Coordinador:

Por este medio tengo el gusto de informar a usted, que he concluido con el asesoramiento del trabajo de graduación con título: **Diseño e implementación de un sistema de adquisición de datos para un detector GRB en Guatemala**, desarrollado por el estudiante *Luis Guillermo García Ordóñez*, con carné 200714819. Después de revisar su contenido final doy mi entera aprobación al mismo.

Atentamente,

Walter Giovanni Alvarez Marroquín
Ing. Electricista
Colegiado No. 8113

Ing. Walter Giovanni Álvarez Marroquín
Colegiado No. 8113

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

Ref. EIME 14. 2015
Guatemala, 11 de febrero 2015.

Señor Director
Ing. Guillermo Antonio Puente Romero
Escuela de Ingenierìa Mecànica Elèctrica
Facultad de Ingenierìa, USAC.

Señor Director:

Me permito dar aprobación al trabajo de Graduación titulado: DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE ADQUISICIÓN DE DATOS PARA UN DETECTOR GRB EN GUATEMALA, del estudiante, Luis Guillermo García Ordóñez, que cumple con los requisitos establecidos para tal fin.

Sin otro particular, aprovecho la oportunidad para saludarle.

Atentamente,
ID Y ENSEÑAD A TODOS

Ing. Carlos Eduardo Guzmán Salazar
Coordinador Area Electrónica

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

STO

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

REF. EIME 14. 2015.

El Director de la Escuela de Ingeniería Mecánica Eléctrica, después de conocer el dictamen del Asesor, con el Visto Bueno del Coordinador de Área, al trabajo de Graduación del estudiante; LUIS GUILLERMO GARCÍA ORDÓÑEZ titulado: DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE ADQUISICIÓN DE DATOS PARA UN DETECTOR GRB EN GUATEMALA, procede a la autorización del mismo.

Ing. Guillermo Antonio Puente Romero

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

GUATEMALA, 12 DE MARZO 2,015.

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

DTG. 369.2015

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Eléctrica, al Trabajo de Graduación titulado: **DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE ADQUISICIÓN DE DATOS PARA UN DETECTOR GRB EN GUATEMALA,** presentado por el estudiante universitario: **Luis Guillermo García Ordoñez,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

Ing. Pedro Antonio Aguilar Polanco
Decano

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

Guatemala, 29 de julio de 2015

/gdech

ACTO QUE DEDICO A:

Dios	Por permitirnos hacer ingeniería inversa del mundo que nos rodea.
Mis padres	María Elena Ordóñez de García y Miguel Ángel García de León, por su apoyo incondicional en todo momento de mi vida, sin ellos no habría llegado hasta aquí.
Mi hermana	Por ser un ejemplo de humildad, valentía y perseverancia, a ti todo mi amor y admiración.
Mi sobrina	La niña que ya creció y que está tomando su propio camino, el camino es largo, pero gratificante.
Mi familia	Por todo el apoyo que me han brindado y por estar presente en cada paso de mi vida, y muy especialmente a mi tío Mario Adolfo Ordóñez Marroquín, por enseñarme que en la vida no todo son libros, pero qué triste sería todo sin ellos.
Mi novia y amigos	De estudios, del trabajo, de la vida, los que tengo, tuve y tendré. Sin ustedes, el camino hasta aquí sería muy aburrido.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por ser el faro de conocimiento para miles de guatemaltecos que buscan crecer y superarse día a día.
Departamento de Física, Laboratorio de Electrónica	Por aceptarme como auxiliar, por apreciarme como persona y por ser parte importante de mi desarrollo profesional.
Proyecto LAGO	Por permitirme desarrollar este proyecto y por abrirme las puertas a un campo completamente nuevo. Agradecimiento especial a Miguel Sofo y a Rubén Conde, por su asesoría en los momentos justos. A Iván Morales y Maynor Ballina, con quienes iniciamos el proyecto sin tener idea de lo que nos esperaba.
Mi asesor	Ingeniero Walter Álvarez, por motivarme a tomar este camino, por lo cual le estaré siempre agradecido.
La ENCA	Por ser el primer peldaño de mi formación, no olvido de donde vengo para tener claro a donde voy.

Las instituciones Que apoyaron el proyecto por medio de muestras gratis y donaciones de las cuales puedo mencionar: Texas Instruments, Maxim IC, TE Connectiviy, Xilinx, Altera, ICTP; en especial a su director Fernando Quevedo y a los miembros del MLAB.

Mi novia Jacqueline Moran, por toda la paciencia y tolerancia que conlleva ser la novia de un estudiante de electrónica. Por los momentos buenos y malos que vivimos en estos años. Nunca lo olvidaré.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
AC	Corriente alterna
DC	Corriente directa
Hz	Hertz
Z	Impedancia
μ	Micro
ms	Milisegundo
ns	Nanosegundo
Ω	Ohmio
PCB	*Printed circuit board*
SPS	*Samples per second*

GLOSARIO

BIT Unidad mínima de información en comunicaciones informáticas.

Break out board Hardware que permite el fácil acceso a un microchip por medio de pines.

Caja blanca Sistema de pruebas donde se tiene acceso a cada uno de los componentes del sistema permitiendo observar los valores de las señales internas.

Caja negra Sistema de pruebas donde no se analizan los componentes internos del sistema analizando únicamente la respuesta a diversas señales de entrada llamadas estímulos.

Canal Medio de transmisión por el que viajan las señales portadoras de información emisor y receptor.

Ciclo de trabajo Tiempo que permanece en alto una señal generada por un PWM.

Conversión A/D Conversión análoga a digital, consiste en la transcripción de señales analógicas en señales digitales, con el propósito de facilitar su procesamiento.

Conversión D/A Conversión digital a análoga. Es el proceso inverso de la conversión A/D. Se parte de muestras en formato digital (valores discretos), y estas se deben convertir en una señal analógica (valores continuos).

Corriente oscura Toda la corriente que se genera por los electrones que se desprenden dado un exceso de voltaje en los dínodos.

Crosstalking Efecto de inducción magnética mutua entre dos pistas paralelas en una PCB.

Debugging Proceso metódico utilizado para detectar y corregir errores (*bugs*) y defectos dentro de un software de computadora o una pieza de hardware.

Dínodo Nombre que reciben cada uno de los electrodos de un tubo fotomultiplicador. Cada dínodo está cargado más positivamente unos 100 voltios.

Efecto fotoeléctrico Emisión de electrones por un material cuando se hace incidir sobre él una radiación electromagnética. A veces se incluyen en el término otros tipos de interacción entre la luz y la materia.

Eficiencia cuántica Cantidad definida para un dispositivo fotosensible como la película fotográfica o un CCD como el porcentaje de fotones que chocan con la superficie fotorreactiva que producirá un par electrón-hueco.

ESR Resistencia equivalente en serie, (*equivalent series resistor*). Es la parte resistiva de la impedancia de cierto componente eléctrico.

Firmware Bloque de instrucciones de máquina para propósitos específicos, grabado en una memoria, normalmente de lectura/escritura.

Footprint Esquema con las medidas de uno o varios circuitos integrados de montaje superficial a traspasarse a una lámina de cobre previo al grabado.

Front-end electronics La electrónica para realizar la adquisición de datos presente en un WCD.

Hardware Conjunto de componentes físicos de un sistema.

Jumper Puente variable en un PCB utilizado para seleccionar una determinada entrada o salida dentro de un circuito.

Layout	En PCBs, es el esquema de distribución de elementos, así como las rutas de interconección de los mismos.
Máxima transferencia energética	Establece que, dada una fuente, con una resistencia de fuente fijada de antemano, la resistencia de carga que maximiza la transferencia de potencia es aquella con un valor óhmico igual a la resistencia de fuente.
Osiloscopio	Instrumento electrónico que permite la visualización de señales eléctricas en una escala temporal.
Overdrive	Bit que indica cuando un valor se sale de la escala de medición.
PCB	Placa de circuito impreso, (Printed circuit board). Soporte mecánico y eléctrico que conecta todos los componentes electrónicos por medio de hojas de cobre grabadas.
PLD	Dispositivo lógico programable, por sus siglas en inglés; es un tipo de diseño implementado en chips que permite la reconfiguración de los circuitos con el simple cambio del software que incorpora.

PMT	Tubo fotomultiplicador o *photomultiplier tube*, por sus siglas en inglés; es un transductor sensible a la luz la cual la convierte a corriente eléctrica, la cual es amplificada por medio de dínodos
Polarización electromagnética	Propiedad de las ondas electromagnéticas que pueden oscilar con más de una orientación.
Radiación Cherenkov	Radiación de tipo electromagnético, producida por el paso de partículas en un medio a velocidades superiores a las de la luz en dicho medio.
Regulador LDO	Regulador de baja caída, por sus siglas en inglés; es un regulador de voltaje lineal que opera con un voltaje diferencial bajo de entrada y salida por lo que es de alta eficiencia.
Ruido eléctrico	Señal electromagnética, ajena a la medición, generada por la corriente inducida en el cableado eléctrico.
Saturación	Estado de un circuito amplificador que suministra una tensión de salida próxima a la de aquella con la que se alimenta.
Skew	Retraso en el tiempo de llegada entre los pines de una salida paralela.

SPI	Interfaz serial periférica, (Serial Peripheral Interface). es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos.
SPICE	Programa de simulación con énfasis en circuitos integrados, Simulation Program with Integrated Circuits Emphasis.
SPS	Muestras por segundo (Samples per Second). Número de muestras que obtiene un dispositivo en cada segundo.
Standby	En circuitos integrados, es el estado de bajo consumo energético a espera de una señal para reanudar el funcionamiento.
Trigger	Función de disparo del osciloscopio, configuración del osciloscopio que permite indicar el nivel mínimo para que capture los datos de interés. Dependiendo de las opciones disponibles, puede configurarse según flanco ascendente o descendente, en cualquiera de los canales analógicos, o de forma externa.
UART	Universal Asynchronous Receiver-Transmitter, es el dispositivo que controla los puertos y dispositivos serie.

VHDL	Lenguaje de descripción de hardware de alta velocidad (Very High Speed Integrated Circuit). Es un lenguaje definido por el IEEE (Institute of Electrical and Electronics Engineers) (ANSI/IEEE 1076-1993) usado por ingenieros para describir circuitos digitales y programar PLD.
WCD	Water Cherenkov Detector, es un detector de partículas altamente energéticas sensible a la radiación Cherenkov.
Zero scale voltage	Voltaje a escala cero. Es el voltaje en la salida de un amplificador diferencial cuando es cero el valor de la entrada.

RESUMEN

En el presente trabajo de graduación se detalla el diseño y la implementación de un sistema de adquisición de datos, para interactuar con un detector de partículas de efecto Cherenkov a utilizarse como parte del proyecto LAGO.

En los primeros capítulos se detalla el objetivo del proyecto LAGO, así como un marco de introducción general de conceptos claves a considerar en el proyecto. Además; se describe la electrónica implementada anteriormente, haciendo un recuento de la evolución de los componentes utilizados en cada versión y el trabajo de otras delegaciones.

Posteriormente se indica el proceso de diseño de la nueva electrónica, describiendo las especificaciones de diseño, la selección de los componentes y los criterios de selección para cada bloque.

Con el proceso de diseño se presentan los resultados obtenidos del primer prototipo detallando los errores comunes y las correcciones a realizar, así como las consideraciones adicionales a tomar en el diseño del PCB.

Por último, se detalla el prototipo final con las correcciones realizadas junto con el diseño definitivo, y del PCB utilizado.

OBJETIVOS

General

Diseñar e implementar un sistema de adquisición de datos de alta velocidad y resolución a utilizarse en un detector de destellos de radiación gamma (GRB), como parte del proyecto LAGO-Guatemala

Específicos

1. Dar a conocer el propósito del proyecto LAGO y la electrónica de adquisición de datos utilizadas para el conteo de partículas generadas durante un GRB.

2. Presentar los antecedentes de las electrónicas implementadas y la justificación para el diseño de una nueva.

3. Describir las especificaciones de la nueva electrónica de adquisición de datos a utilizarse en el proyecto LAGO-Guatemala.

4. Dar a conocer las condiciones de implementación del prototipo previo al diseño definitivo.

5. Detallar el proceso de corrección de errores en el prototipo y presentar los cambios realizados en el diseño final.

6. Describir las pruebas realizadas al diseño definitivo para comprobar su correcto funcionamiento.

INTRODUCCIÓN

La física de astropartículas es una rama de investigación relativamente reciente. Se dedica al estudio de partículas elementales de origen astrofísico. Los rayos cósmicos y rayos gamma de altas energías son parte de los llamados cinco pilares de la física de astropartículas. Sin embargo, estas tienen un tiempo de vida muy corto dentro de la atmósfera, lo que dificulta su estudio.

Para realizar la detección de estos eventos es necesaria la electrónica de alta velocidad que pueda adquirir información para catalogarlos. Sin embargo, muchas veces el costo de esta electrónica es privativo para países como Guatemala, en donde el presupuesto para proyectos de investigación es limitado.

A pesar de esta situación, muchas instituciones y entidades contribuyen con el desarrollo de la investigación en Guatemala por medio de donaciones y muestras gratis, las cuales permiten que se lleven a cabo proyectos de bajo presupuesto. Uno de estos es el proyecto LAGO, el cual estudia destellos de radiación gamma por medio de detectores Cherenkov a lo largo de Latinoamérica. Estos detectores son de bajo presupuesto y se construyen, en su mayoría, con piezas recicladas del observatorio Pierre Auger. Uno de estos detectores fue donado por laboratorio de detección de partículas del Centro Atómico de Bariloche, Argentina, el cual, desafortunadamente, quedó en desuso por una falla en la electrónica de adquisición de datos.

Ante esta falla se estudió la posibilidad de reparar la electrónica dañada y actualizar la misma, diseñando una tarjeta de bajo costo y mejor resolución. Con

esta idea se inicia el proceso de diseño de la nueva electrónica de adquisición de datos, buscando componentes de alta velocidad y bajo costo, intentando mantener cierto grado de retrocompatibilidad con las electrónicas implementadas en el pasado.

Además de presentar el diseño de la nueva electrónica se señalan conceptos a considerar en las posteriores actualizaciones del diseño, presentando la actual electrónica como una base para futuras contribuciones, por parte de la delegación LAGO-Guatemala, hacia la electrónica de adquisición, pretendiendo abrir una puerta al desarrollo de electrónica de instrumentación en Guatemala y crear la oportunidad de trabajar en conjunto con otras disciplinas, como la física experimental, para el desarrollo de un proyecto en común.

1. GRB Y PROYECTO LAGO

A finales de la década de los sesenta, producto de la Guerra Fría, dos satélites estadounidenses enviados para detectar pulsos de radiación gamma producidos por pruebas de bombas atómicas en el espacio, detectaron un destello de radiación gamma distinto al emitido por cualquier bomba atómica conocida.

Luego de analizar los datos recolectados por los satélites y hacer la correlación de tiempos, se descubrió que los destellos provenían de fuentes fuera de la tierra. Estos eventos fueron denominados: brotes de rayos gamma o GRB, por sus siglas en inglés.

El descubrimiento de los GRB dio inicio al desarrollo de instrumentación para medir estos eventos y así estudiar el origen y causas de los mismos. En el presente capítulo se detalla brevemente qué son estos destellos, así como los antecedentes que han llevado hasta el proyecto LAGO (Large Aperture GRB Observatory), para el cual se ha desarrollado esta instrumentación.

1.1. Brotes de radiación gamma

Los GRB son destellos de rayos gamma asociados a explosiones extremadamente energéticas observadas en galaxias distantes. Son los eventos electromagnéticos más brillantes conocidos en el universo.

La mayoría de los GRB observados se cree que consisten en un angosto haz de radiación intensa liberado de una supernova cuando una estrella masiva de rápida rotación colapsa para formar una estrella de neutrones, estrella de *quarks* o un agujero negro. Una subclase de GRB (denominados rayos cortos) aparentemente se originan de la fusión de estrellas binarias de neutrones.

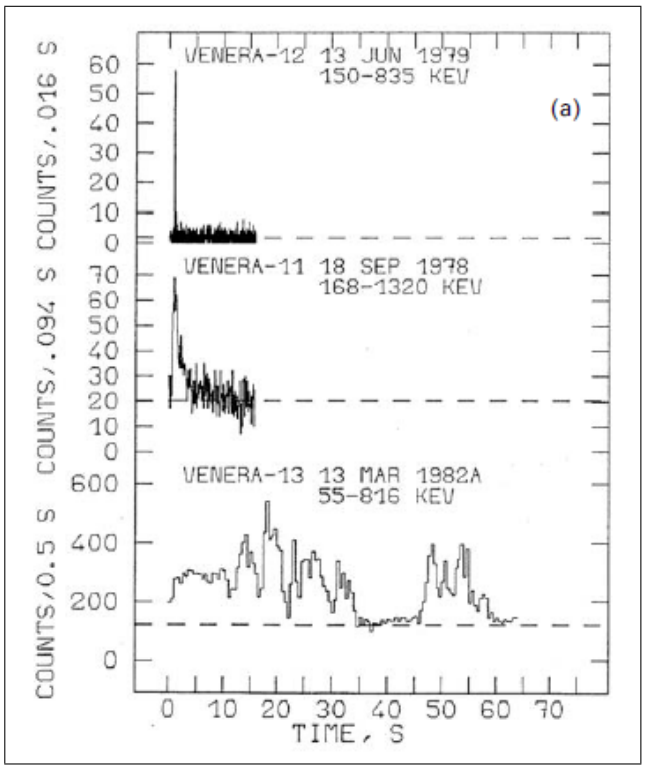
La mayoría de las fuentes de GRB se encuentran a billones de años luz de la tierra, implicando que las explosiones son extremadamente energéticas y raras. Como medida de comparación se puede decir que un destello típico libera tanta energía en pocos segundos como la energía que libera el sol en sus 10 billones de años de tiempo de vida.

Muy pronto se descubrió que las curvas de luz de los GRB eran variables de un destello a otro. Tanto las formas como las escalas temporales varían de unos milisegundos a cientos de segundos. Los perfiles de tiempo pueden presentar múltiples picos como se observa en la figura 1.

Esta variación de formas se ha utilizado para clasificar tentativamente los GRB en eventos largos (duración aproximada de dos segundos) y cortos (menor a dos segundos generalmente menores a 0,3 segundos).

Debido a la gran cantidad de energía que se libera en cada destello se han propuesto diversas hipótesis del origen de los mismos. La interpretación actual corresponde a una gran cantidad de energía gravitatoria que se libera en corto tiempo (segundos o menos) en una pequeña región del espacio (10 kilómetros o menos) en un evento cataclísmico estelar. Parte de esta energía se escapa en los primeros segundos en forma de neutrinos térmicos, otra fracción en forma de ondas gravitacionales.

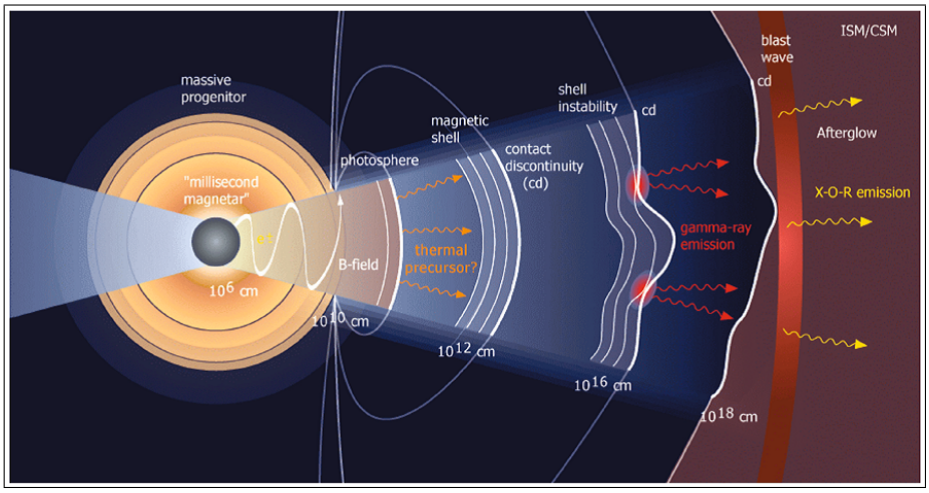
Figura 1. **Curvas de luz de GRB, muestras de diversidad de eventos**



Fuente: VEDRENNE, Gilbert; ATTEIA, Jean-Luc. *Gamma-Ray Burst, the brightest explosions in the Universe*. p. 3.

Esta repentina liberación de energía podría resultar en una bola de fuego de altas temperaturas expandiéndose a velocidades relativistas, sufriendo así disipación interna la cual conduce a rayos gamma. Este efecto se observa en la figura 2

Figura 2. **Ilustración de la producción de rayos gamma (γ)**



Fuente: VEDRENNE, Gilbert; ATTEIA, Jean-Luc, *Gamma-Ray Burst, the brightest explosions in the Universe*. p. 609.

1.2. Detectores de radiación gamma

Cuando se descubrieron los GRB, los satélites Vela no estaban equipados para realizar una espectroscopia de los eventos gamma. Rápidamente se enviaron otras misiones espaciales para determinar de donde provenían estos eventos sin mucho éxito.

En 1991, cuando la NASA envió una misión espacial llamada Compton Gamma-Ray Obserbatory (Observatorio Compton de rayos gamma) con un detector extremadamente sensible llamado BATSE por sus siglas en inglés. Este instrumento proveyó el primer gran juego de datos cruciales sobre los GRB indicando que su patrón de radiación es isotrópico.

Los datos que BATSE proporcionó permitieron un avance sobre el origen de los GRB, esto dio inicio a una serie de misiones espaciales como el satélite italiano-holandés BEPPOSAX, el cual contaba con instrumentación más especializada para la detección de GRB.

BEPPOSAX contenía cinco instrumentos:

- Espectrómetro concentrador de baja energía
- Espectrómetro concentrador de energía media
- Contador proporcional centellador de gas de alta presión
- Sistema detector de Phoswich
- Cámara de campo ancho

BEPPOSAX funcionó hasta el 2002 y BATSE fue sacado de órbita en el 2000. Sin embargo, la revolución en el estudio de los GRB motivó el desarrollo de instrumentación adicional diseñada para explorar la naturaleza de los GRB específicamente en los momentos iniciales seguidos de la explosión.

Para esa misión se envió a HETE-2 (2000) y más adelante a SWIFT (2004). El primero logró descubrir la conexión de los GRB con las supernovas, entre otros descubrimientos importantes. El segundo, que aún se encuentra en operación, está equipado con un detector de rayos gamma ultra sensible, así como con telescopios de rayos X y ópticos, los cuales pueden ser basculados para observar el brillo emitido que sigue al destello gamma.

A pesar del éxito de las misiones espaciales, el costo es elevado. Cada misión espacial involucra una inversión millonaria seguida por el hecho de que únicamente se puede cubrir una región del cielo a la vez. Estas limitantes han motivado el desarrollo de detectores de GRB's en tierra, utilizando la técnica de conteo de una sola partícula por medio de detectores Cherenkov de agua.

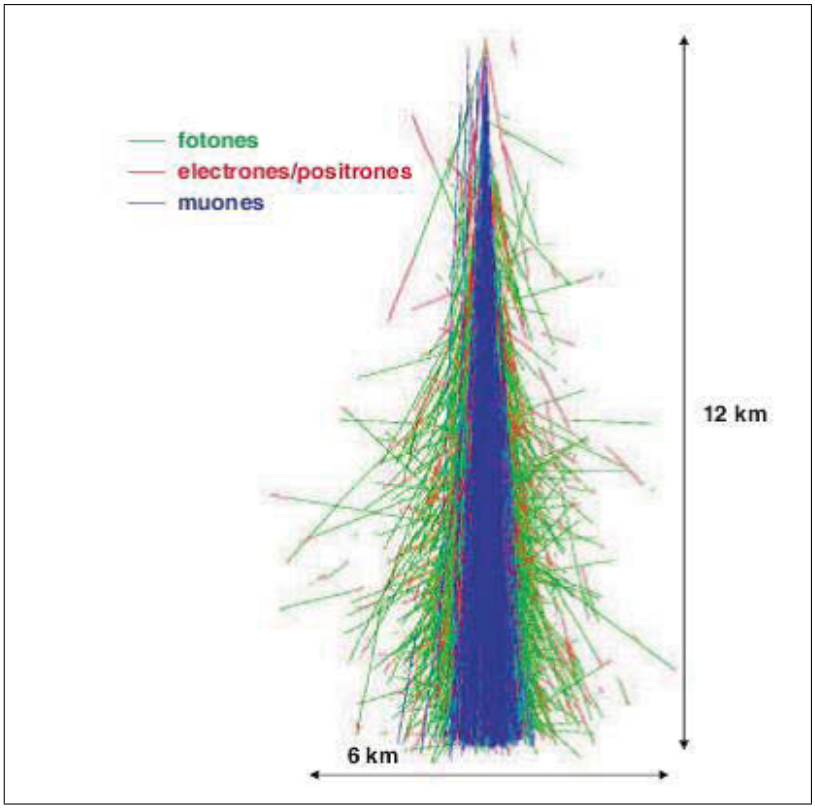
1.2.1. Técnica de una sola partícula

Cuando los fotones altamente energéticos alcanzan la atmósfera, producen una cascada de rayos cósmicos que pueden ser detectados. Las energías siguen siendo bajas aún para ser detectados a nivel del suelo. Sin embargo, se espera que un buen número de estos fotones lleguen durante el destello en un periodo corto de tiempo (típicamente 1 segundo) y que algunos logren golpear en un detector a nivel del suelo. La figura 3 muestra un ejemplo de la dispersión de un fotón al momento de ingresar a la atmósfera.

Aprovechando este comportamiento se han creado observatorios como Pierre Auger, el cual es una iniciativa en conjunto de más de 20 paises, ubicado a 1 400 msnm en un área de 3 000 kilómetros cuadrados con el propósito de estudiar la radiación cósmica que llega a la Tierra.

El número de partículas detectadas en tierra es proporcional a su altitud. A razón de esto, se han realizado diversos experimentos como INCA, ubicado en Chacaltaya, Bolivia (5 200 msnm); Argo en el Tibet (4 300 msnm); LAGO (diversas locaciones); los cuales compensan la extensión de tierra con altitud.

Figura 3. **Ejemplo de una lluvia de partículas**



Fuente: PÉREZ, Yunior. *Caracterización de Detectores Cherenkov en el Proyecto LAGO*. p. 21.

1.3. Efecto Cherenkov

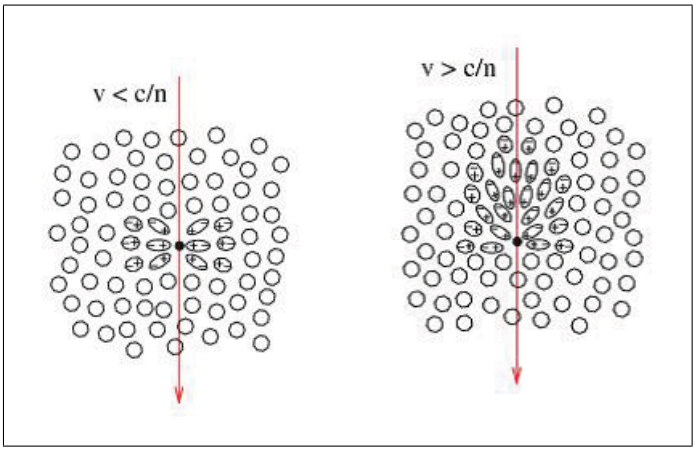
La radiación de Cherenkov es de tipo electromagnético producida por el paso de partículas en un medio a velocidades superiores a la de la luz en dicho medio.

La radiación Cherenkov es un tipo de onda de choque que produce un brillo azulado. Este es un fenómeno similar al de la generación de una onda de choque cuando se supera la velocidad del sonido.

Debido a las propiedades ondulatorias de la luz, se pueden producir los mismos efectos que con las ondas mecánicas. Los frentes de onda esféricos se superponen y forman uno solo en forma cónica. Esto ocurre cuando las partículas viajan a velocidades superiores a los fotones en dicho medio.

Para que se produzca el efecto Cherenkov, la partícula que atraviesa el medio debe tener carga eléctrica y el medio debe ser dieléctrico para que sea afectado por la carga de la partícula.

Figura 4. **Comparación entre dos partículas cargadas a distintas velocidades en un medio dieléctrico**



Fuente: PÉREZ, Yunior. *Caracterización de Detectores Cherenkov en el Proyecto LAGO*. p. 28.

En la región cercana al paso de la partícula, los átomos que componen el material se distorsionan polarizando el medio alrededor de la partícula. Si la partícula se mueve rápidamente, el campo de polarización no es completamente simétrico a lo largo del eje.

Este efecto crea un campo momentáneamente por la partícula en cada elemento a lo largo de su trayectoria. Si la velocidad de la partícula es mayor a la de la luz en el medio, es posible que todas las ondas del campo formado se encuentren en fase unas con otras emitiendo fotones.

1.3.1. Detector Cherenkov

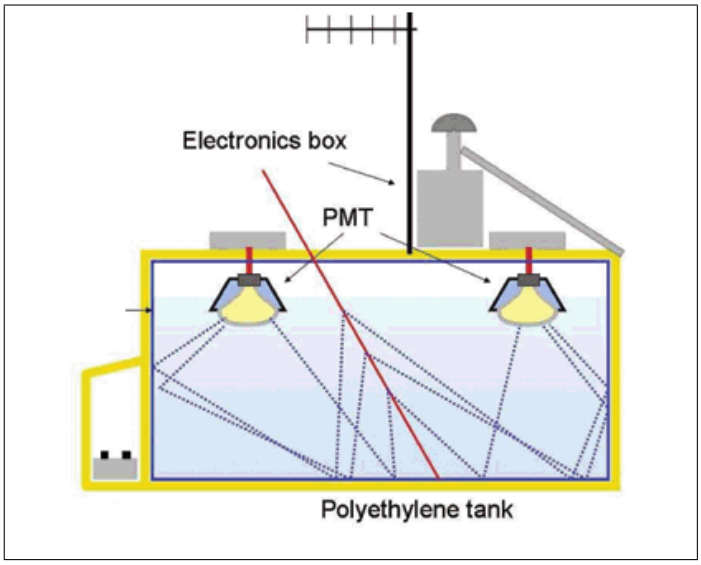
Es un detector de partículas que utiliza el principio de la radiación Cherenkov para detectar partículas altamente energéticas.

La mayoría de los detectores Cherenkov están diseñados para la detección de una partícula primaria cargada y su luz emitida. Algunos detectores pueden determinar algunas partículas secundarias como un evento en una lluvia de partículas. La radiación emitida no siempre se encuentra en el rango visible de luz, la mayoría de veces se encuentra en el rango de la luz ultravioleta. Esta emisión ocurre de manera instantánea al momento de que una partícula cargada pase por el medio (usualmente agua destilada).

El esquema básico de un detector Cherenkov se observa en la figura 5, consiste en un tanque cerrado de forma cilíndrica, completamente sellado de forma tal que no ingrese la luz externa.

En su interior se encuentra revestido por un material reflejante y lleno de agua (bajo algún tratamiento purificador). En la parte central superior del tanque se encuentra un tubo fotomultiplicador (PMT) el cual realiza el conteo de fotones previo a su digitalización.

Figura 5. **Diagrama básico de un detector Cherenkov**



Fuente: CINTRA, Ronald. *First results from the Pierre Auger Observatory* [en línea] http://www.scielo.br/img/revistas/bjp/v36n4a/a12fig02.gif. Consulta: noviembre de 2013

1.4. Proyecto LAGO (Large Aperture GRB Observatory)

Es un proyecto internacional que tiene como objetivo la detección de destellos de rayos gamma usando detectores Cherenkov situados en altura en diversos países.

Según el documento: *The Large Aperture GRB Observatory*, la absorción en la atmósfera es tal que a 1 400 msnm (la altura del observatorio Auger), la cantidad de partículas es 100 veces menor que a 5 200 m. (Chacaltaya) y el ruido es aproximadamente 8 veces menor.

Un detector Cherenkov tipo Auger colocado en Chacaltaya sería por lo tanto equivalente a $(100\sqrt{8})^2 = 1\,250$ detectores a la altura de Auger. Con 20 metros cuadrados de detectores a 5 200 metros de altura se obtiene un umbral menor al de 1600 detectores de Auger, sin tomar en cuenta las mejoras provenientes de la frecuencia de toma de datos.

El proyecto LAGO se propone mejorar el potencial de detección de los actuales observatorios usando detectores Cherenkov de agua en alta montaña.

Guatemala ingresó al proyecto LAGO en el 2011, ubicando un tanque prototipo a una altura de 1 490 msnm en una latitud de 14º/35'/18,06"/N y 90º/33'/13,30"/W. El tanque se mantuvo en funcionamiento hasta marzo de 2012, en el cual la electrónica sufrió de un desperfecto.

2. ELECTRÓNICA DEL PROYECTO LAGO

Como se mencionó en el capítulo anterior, el proyecto LAGO utiliza detectores de agua Cherenkov para la detección de las partículas. Un detector Cherenkov convencional puede detectar partículas con energías desde 10 GeV hasta el orden de los TeV. La electrónica presente en el detector tiene la tarea de detectar los fotónes emitidos, transformar la señal lumínica en impulsos eléctricos, amplificar la señal, digitalizarla y enviarla a una base de datos para su posterior análisis.

En el presente capítulo se detalla la electrónica del detector Cherenkov así como las características deseables de cada módulo.

2.1. Características deseables en la electrónica de un detector de partículas

A pesar de los diversos tipos de dispositivos electrónicos que se encuentran actualmente y a pesar de los diversos diseños y prototipos que se han desarrollado para la detección de partículas, los principios básicos se conservan.

El propósito de la electrónica de diseño (*front-end*) y los sistemas de procesamiento de señales según se cita en el libro *Particle Detectors*, es:

- Adquirir una señal eléctrica del sensor
- Ajustar el tiempo de respuesta del sistema para optimizar:

- La mínima señal detectable
- La medición de energía
- Tasa de eventos
- Tiempo de llegada (medición de tiempo)
- Insensividad a la forma de pulso del sensor
- Alguna combinación de lo anterior

- Digitalizar la señal y almacenar los datos para su posterior análisis

Sumado a lo anterior, existen otras características que hay que tomar en cuenta al momento de diseñar la electrónica de los detectores de partículas como las condiciones ambientales, el consumo energético de la electrónica, el valor económico, entre otros.

Generalmente no se puede optimizar todo lo expuesto anteriormente, por lo que un buen diseño debe considerar qué características son prioritarias y cuales deben ser sacrificadas para la fabricación del mismo.

2.2. Componentes de la electrónica LAGO

La electrónica para realizar la adquisición de datos presente en un WCD debe ser capaz de capturar los fotones emitidos durante un evento Cherenkov, convertirlos a una señal analógica, digitalizarlos y transmitirlos para su posterior análisis.

La electrónica dentro de un WCD se resume en:

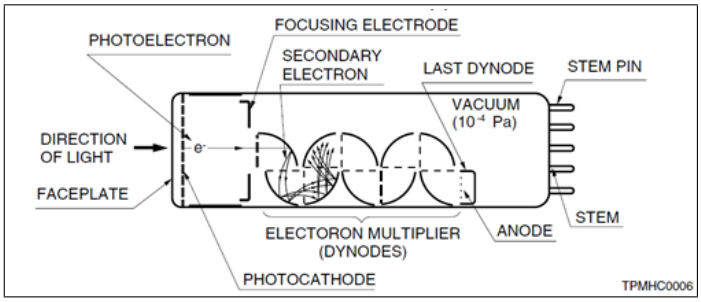
- Tubo fotomultiplicador (PMT)
- Interfáz de adquisición de datos

- Unidad de control y transmisión de datos (UCT)

2.2.1. Tubo fotomultiplicador (PMT)

Es un dispositivo versátil que provee una respuesta ”ultra-rápida” y extrema sensibilidad ante eventos de respuesta lumínica. Un típico PMT consiste en un cátodo fotoemisivo (fotocátodo) seguido de electrodos de enfoque, un multiplicador de electrones (dinodos), y un colector (ánodo). En la figura 6 se muestran de manera simplificada dichas partes.

Figura 6. **Descripción gráfica de un tubo fotomultiplicador**



Fuente: HAMAMATSU. *Photomultiplier Tubes*. p. 10.

El principio del funcionamiento de un PMT se basa en el efecto fotoeléctrico. Los fotones incidentes chocan con un plato fotosensible, el cual libera una cantidad proporcional de electrones. Estos electrones chocan a su vez contra los dínodos. El alto voltaje entre los dínodos provoca una avalancha de electrones en cada colisión multiplicando la señal.

El flujo de los electrones desplazados generan una corriente que es directamente proporcional al número de fotones. Conociendo la ganancia del PMT y su eficiencia cuántica se puede realizar un conteo de fotones incidentes.

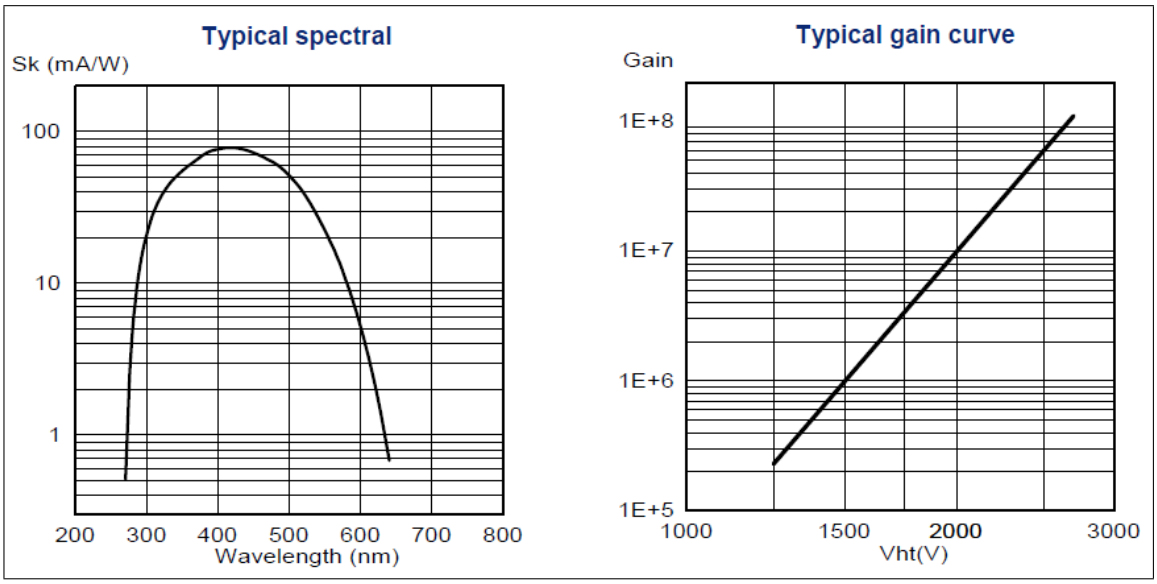
El proyecto LAGO-Guatemala cuenta con un PMT modelo XP1802 el cual tiene las siguientes caracteristicas:

- Rango de detección: 270 a 650 nm
- Máximo de detección: 420 nm
- Tiempo de subida: 3 ns
- Tiempo de tránsito: 1.4 ns
- Impedancia de salida: 50Ω

La curva de respuesta se muestra en la figura 7, en la cual se observa la respuesta espectral y la curva de ganancia. La ganancia está en orden de 10^6 a 10^8 y tiene un rango lineal aproximado de 1 200 voltios hasta 3 000 voltios.

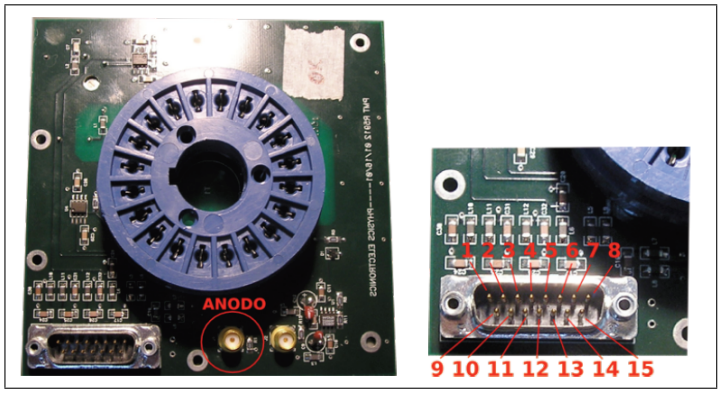
Para su funcionamiento necesita un voltaje de alimentación de 2 000 voltios uniformemente distribuido entre los dínodos. Un cambio drástico en el voltaje puede generar el desprendimiento aleatorio de electrones en los dínodos generando ruido. [1] Para controlar esto el PMT cuenta con una placa de control la cual tiene la tarea de controlar el aumento y la estabilidad del voltaje de una manera gradual, así como un acople coaxial SMA para la transmisión de datos.

Figura 7. **Respuesta espectral y curva de ganancia del PMT XP1802**



Fuente: PHOTONIS. *XP1802 Datasheet*. p. 3.

Figura 8. **Placa de control del PMT**



Fuente: Centro Atómico Bariloche. *TN LAGO-2011_004 Bariloche*. p. 23.

2.2.1.1. Control del PMT

La placa de control del PMT fue donada por la comunidad LAGO-Bariloche y se muestra en la figura 8 junto con un conector DA-15 utilizado para la interfaz. Para el control de dicha placa se necesitan cuatro valores de voltaje para alimentarlos.

- $+3,3V$
- $-3,3V$
- $+5,0V$
- $+12,0V$

Ubicados en los pines 1 al 4 respectivamente. El propósito de estos valores de voltaje se explica más adelante.

Los demás pines son $V_{Control}$ (pin 5), tierra (pines 9 al 13) y dos pines de interfaz con un transductor de temperatura AD592 (pines 8 y 15), el cual ya viene incorporado en la placa, el resto son no conectados.

El $V_{Control}$ tiene la tarea de controlar, por medio de una señal analógica de 0 a 5 voltios el umbral de voltaje con una relación típica de $1:400$. Es decir, por cada 0,25 voltios que aumenten en el $V_{Control}$ son 100 voltios que aumentan en el PMT.

Para simplificar la interfáz la placa tiene un conversor DA-15 a RJ-45. El *pinout* se detalla en la tabla I

[1]Otro efecto indeseable de este desprendimiento se conoce como corriente oscura y ocurre con las variaciones de voltaje y temperatura que provocan una lectura de corriente en orden de miliamperios.

Tabla I. ***Pinout* de salida de la placa de control del PMT**

Pines	**DA-15**	**RJ-45**
1	+3,3V	+3,3V
2	-3,3V	-3,3V
3	+5,0V	+5,0V
4	+12,0V	GND
5	$V_{Control}$	$V_{Control}$
6	NC	pin 3 AD592
7	NC	pin 1 AD592
8	pin3 AD592	NE
9	GND	NE
10	GND	NE
11	GND	NE
12	GND	NE
13	GND	NE
14	NC	NE
15	pin1 AD592	NE

Fuente: elaboración propia.

Todo el hardware descrito anteriormente fue donado por la delegación LAGO-Bariloche. Toda la información del acople de hardware se encuentra en la nota técnica *LAGO Official Electronics guide* en el documento, *Guía de conexión de hardware*. Como planes futuros se tiene el diseño de un nuevo hardware de acople del PMT desarrollado por la delegación Guatemala.

2.2.2. Interfaz de adquisición de datos

La señal analógica proveniente del PMT debe ser procesada antes de analizar los datos. La electrónica que recibe de frente la señal analógica proveniente del sensor se le conoce como *front-end*. Esta electrónica tiene la tarea de recibir la señal análoga, darle forma, acoplarla y digitalizarla previo a su envío a una base de datos.

Además de proporcionar un camino a la señal de entrada, dado que esta interfaz tiene un camino directo entre el sensor, tiene la tarea de acoplar las señales provenientes de la unidad de control hacia el control de voltaje en el PMT.

La interfaz de adquisición de datos consta de las siguientes partes:

- *Pulse shaper*
- *Slow control*
- *Baseline control*
- *Voltage manager*

2.2.2.1. *Pulse shaper*

La señal proveniente del PMT es una curva de corriente negativa. Esta debe ser adecuada previo a su digitalización. El *pulse shaper* tiene la tarea de realizar el acople de impedancias, invertir la polaridad de la señal, y convertirla en una de voltaje con un nivel de tolerancia en el rango del conversor A/D (en adelante llamado ADC, por sus siglas en inglés).

Un *pulse shaper* ideal realiza esta tarea en tiempo real, con un umbral de ruido mínimo y realizando un acople de impedancias que cumpla con el teorema de máxima transferencia de energía.

El ADC utilizado debe ser de alta velocidad, ya que los eventos ocurren en una escala temporal en el orden de 10^{-9} segundos. Para esta tarea se utilizan FADC (Flash Analog to Digital Converters), los cuales permiten tasas de conversión superiores a los 40 MSPS.

2.2.2.2. *Slow control*

El *slow control* o control lento es la parte encargada de enviar al PMT la información del nivel de voltaje de polarización. Dado que el control de voltaje de polarización es realizado por una señal analógica, y la unidad de control trabaja únicamente con señales digitales, el *slow control* realiza la traducción de señales por medio de un conversor D/A (DAC, por sus siglas en inglés)

2.2.2.3. *Baseline control*

Entre las características no ideales del PMT existe una de principal importancia en la adquisición de datos. Esta es el voltaje base (baseline). El PMT tiene un valor base de voltaje en DC que es altamente dependiente de la temperatura.

Si la temperatura ambiente sube o baja, el baseline subirá o bajará también, pudiendo provocar falsos disparos en las lecturas. Para controlar esto se vuelve necesario restar el baseline con un DAC de precisión y un circuito retador análogo. Este control va intimamente ligado con el *pulse shaper*

2.2.2.4. Voltage manager

Como se mencionó en la página 18, el control del PMT necesita entradas específicas de voltaje. El *Voltage manager* tiene la tarea de convertir el voltaje de entrada en los valores requeridos para el control del PMT y de ciertos componentes específicos dentro de la electrónica de adquisición.

2.2.2.5. Características deseadas en la interfaz de adquisición

En las secciones anteriores se explicaron los componentes necesarios para la adquisición de la señal de un PMT; a esto se le llama canal. La interfaz de adquisición puede contar con varios canales según el número de PMTs a usar y la capacidad de la unidad de control de manejar. Además de tener múltiples canales, existen otras características deseables en una interfaz de adquisición. Estas son:

- Rápida respuesta, debe ser capaz de procesar la señal en tiempo real.
- Fácil adquisición, debe ser simple el proceso con una salida fácil de manejar.
- Bajo consumo energético.
- Modular, cualquier falla en un canal no debe interferir con los otros.
- Bajo costo.

2.2.3. Unidad de control y transferencia de datos

Los eventos Cherenkov no ocurren en cada momento, su ocurrencia es aleatoria y no dependen de patrones específicos. Los datos deben ser tomados siempre, sin embargo la salida digital de los datos está en el orden de millones de muestras por segundo por cada canal que se esté usando. Sin embargo, solo son pocos los datos de interés para el estudio de la señal. Esto deja la tarea de discriminar todo lo que sea señal basura.

Para entender mejor el concepto de esta información sobrante se toma como ejemplo el AD9203ARU de 40 MSPS y 10 bits de resolución. Como el muestreo de la señal es permanente tenemos 40 millones de muestras por segundo y cada muestra es de 10 bits. Eso son 400 megabits por segundo de información. En un día esto se convierte en 34,56 terabits de información entrante. Ahora, se asume que en un día ocurra un evento Cherenkov (esto es una exageración con fines ilustrativos, ya que los eventos Cherenkov puede ocurrir uno en meses) de con una duración de 100 nanosegundos y con ayuda del PMT y un una fase de ensanchamiento se hace que la electrónica lo capture en un microsegundo. En ese microsegundo se obtiene, de información, únicamente 40 muestras lo cual serían 400 bits de información. ¡400 bits de información útil de los 34,56 Terabits adquiridos en el día! Si cada bit fuera una gota de agua, se está hablando de aproximadamente una jeringa de 20 mililitros de información útil de 1728 toneladas de agua captadas. Y esto es solo para un canal.

Para manejar este volumen de información se vuelve necesario electrónica de alta velocidad que discrimine en tiempo real los datos útiles y deseche los demás. La mejor opción para instrumentación científica es una FPGA.

El nombre FPGA viene de Field Programable Gate Array y es un dispositivo semiconductor que contiene bloques de lógica y cuya interconexión y funcionalidad puede ser configurada *in situ* mediante descripción de hardware.

Utilizando un FPGA se tiene la facilidad de programar mediante VHDL hardware actual formado por arreglos de unidades lógicas. Esto permite realizar tareas en paralelo de procesos digitales secuenciales o concurrentes. Utilizando estas ventajas se vuelve fácil la tarea de discriminar la señales en tiempo real mientras se controlan los parámetros de voltaje de polarización del PMT y el *baseline control*, así como la transmisión de datos ya discriminados a una PC.

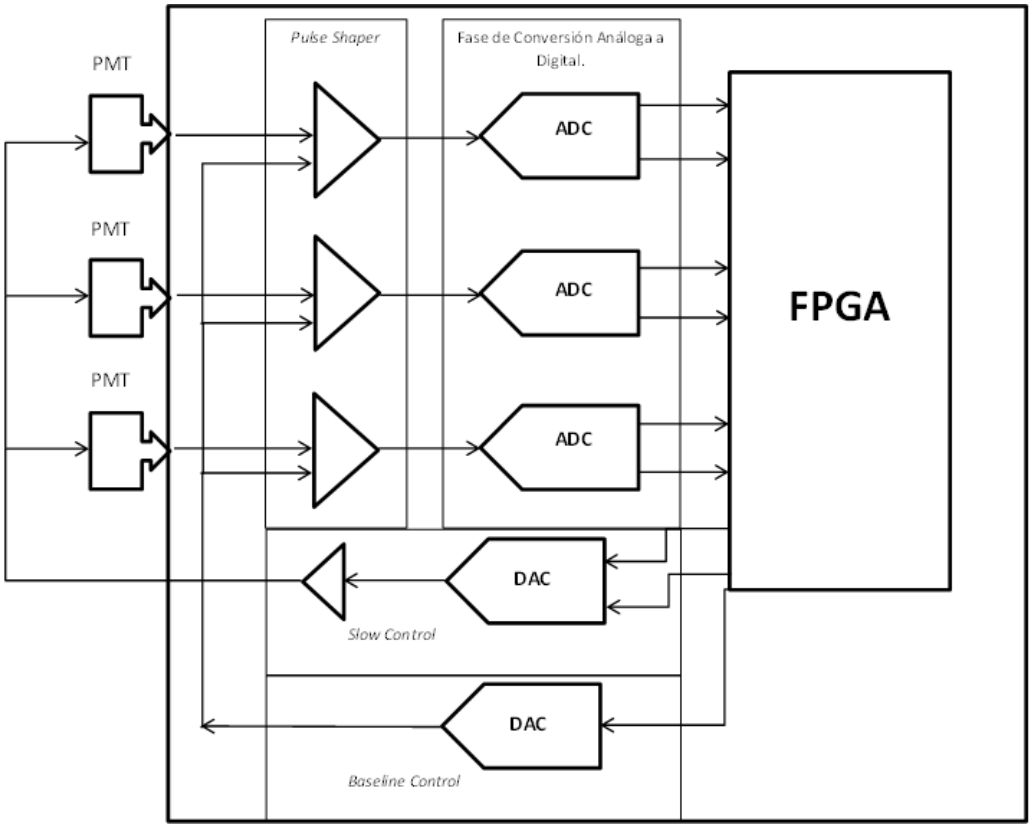
LAGO utiliza oficialmente el FPGA *Spartan 3E* utilizando una tarjeta de desarrollo Nexys 2. Esta tarjeta tiene entre sus caracteristicas 500 kilo-compuertas, un oscilador de 50 Mhz (el cual se puede multiplicar por 4 usando un DCM), un conector *Hirose FX2-100S-1.27DS* con 43 pines de señal para transmisión y recepción de señales a alta velocidad y cuatro conectores P-Mod, todas las señales de entrada y salida tienen protección electrostática y de cortocircuito.

2.3. Electrónicas desarrolladas en el proyecto LAGO

En la figura 9 se muestra el diagrama de bloques de la interfaz de adquisición de datos que se utiliza para la digitalización de los datos.

Durante el desarrollo del proyecto LAGO se han desarrollado diversos sistemas de adquisición de datos buscando mejorar en cada paso las características de sus predecesores.

Figura 9. **Diagrama de bloques de la interfaz de adquisición de datos**



Fuente: elaboración propia.

Dos de las principales tarjetas de adquisición de datos han sido desarrolladas por las delegaciones de Bariloche y México.

2.3.1. Electrónica de Bariloche

Esta electrónica fue desarrollada en el 2011 por Miguel Sofo Haro en el Laboratorio de detección de partículas y radiación del Centro Atómico Bariloche. La tarjeta está diseñada para trabajar con 3 canales simultáneamente. La digitalización ocurre utilizando el dispositivo AD9203ARU de *Analog Devices* a una velocidad de muestreo de 40 MSPS y una resolución de 10 bits.

2.3.2. Electrónica de México

Esta electrónica fue desarrollada en la Benemerita Universidad Autónoma de Puebla (BUAP), por el M.SC. Rubén Conde.

Este sistema se basa en la FPGA NEXYS-2 con un procesador incrustado *microblaze* de 32 Bits creado por Xilinx. La tarjeta desarrollada consta de dos bloques principales conteniendo en cada bloque un ADC dual de 100MSPS. El AD9216 de 10 bits de resolución.

3. DISEÑO DEL SISTEMA DE ADQUISICIÓN DE DATOS PARA LAGO-GUATEMALA: ESPECIFICACIONES Y DISEÑO INICIAL

3.1. El proceso de diseño

El proceso de diseño de una electrónica de adquisición no varía mucho de lo que sería el proceso para un automóvil o un software de computadora. Básicamente consiste en una secuencia de pasos ordenados que dan como resultado un producto final. En la figura 10 se observa un diagrama de flujo que ilustra el proceso de diseño utilizado para el sistema de adquisición de datos.

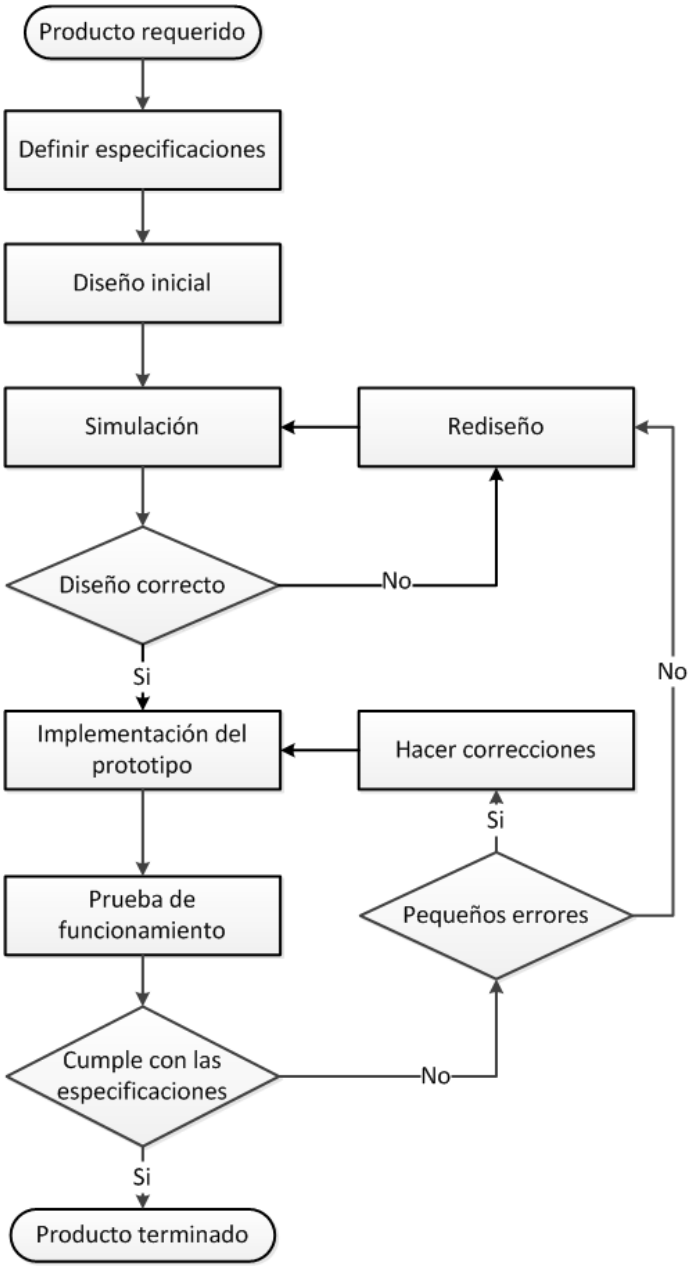
En este capítulo se detalla el diseño inicial, partiendo de las especificaciones requeridas para la electrónica de adquisición. Muchos componentes utilizados no poseen un equivalente en SPICE para realizar la simulación, por lo que se fabricaron Break-out-boards para caracterizar el comportamiento de los mismos.

En el siguiente capítulo se detallan las especificaciones de la tarjeta de adquisición, así como las consideraciones y el diseño inicial.

3.2. Especificaciones del proyecto

A pesar de las ventajas de las electrónicas ya desarrolladas, existen características que se deben optimizar. Principalmente se busca:

Figura 10. **Proceso de diseño**



Fuente: BROWN S. & VRANESIC Z. *Digital Logic with VHDL Design*. p. 7.

- Mayor velocidad de adquisición
- Mayor número de bits
- Menor tamaño

De la teoría de información se conoce que

$$I_k = log_2 M = log_2 \frac{1}{p_k} \tag{3.1.}$$

Donde p_k es la probabilidad de ocurrencia de cada valor. Para un sistema de N bits, sabemos que $p_k = 1/2^N$ por lo que

$$I_k = N$$

Esto indica que: a mayor número de bits, mayor información se puede obtener de la señal analizada.

De forma similar se explica la necesidad de más velocidad de adquisición. El teorema de Nyquist dice:

$$f_{sample} \geq 2 * f_{signal} \tag{3.2.}$$

La ecuación 3.2. indica que la frecuencia mínima de muestreo es el doble de la frecuencia de la señal. En la práctica se necesita una mayor cantidad de muestras para poder adquirir las señales, de hecho, a mayor número de muestras, mejor se puede reconstruir la señal. La razón, menos tiempo de adquisición entre cada muestra. Esto es especialmente útil cuando ocurren eventos simultáneos.

Justificada así la necesidad de cambiar la electrónica de adquisición de datos, se presentan las especificaciones de diseño propuestas para la electrónica LAGO-GT en la tabla II. En las próximas secciones se detalla el diseño inicial de cada bloque de control utilizando con base en las especificaciones descritas.

Tabla II. **Especificaciones de diseño para la electrónica LAGO-GT**

Nombre	Min	Nom	Max	U
Número de bits	-	14	-	Bit
Velocidad de muestreo del ADC	60	-	125	MSPS
Pulse shaper BW	-	125	370	MHz
Impedancia de entrada*	25	50	75	Ω
Ganancia del *pulse shaper**	-6	6	12	dB
Voltage de salida minimo en el *Baseline control*	-1,25	-1	-0,9	V.
Voltage de salida máximo en el *Baseline control*	0,9	1	1,25	V.
Rango de voltaje para el *Slow control*	0	-	5	V.
*Depende de la selección de las resistencias en el amplificador diferencial				

Fuente: elaboración propia.

3.3. Diseño del *pulse shaper*

El *pulse shaper* debe cumplir con las siguientes tareas:

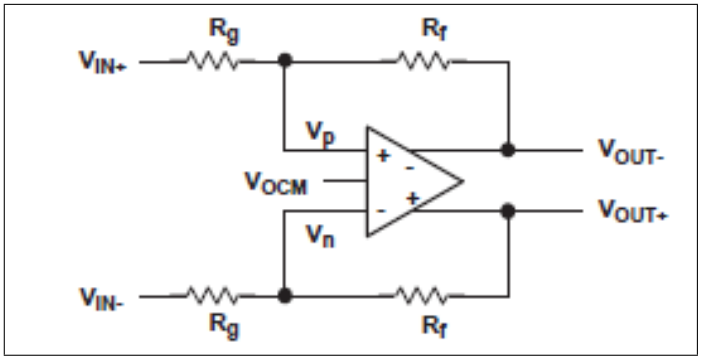
- Acoplar la señal de entrada con una impedancia de entrada de 50Ω.
- Acoplar la señal de entrada al ADC con una señal diferencial de 2,3 voltios de amplitud.
- Realizar la resta con el *baseline control*.

Las tres tareas pueden realizarse utilizando un amplificador diferencial. La elección realizada fue el THS4503 de Texas Instruments. Este integrado tiene una arquitectura completamente diferencial.

Un ancho de banda de 370 Mhz. Salida de modo común (especialmente útil para el acople con el ADC), amplio rango de voltaje de alimentación, alto grado de linealidad.

Aprovechando las características de un amplificador diferencial, este integrado permite realizar las tres tareas en una sola etapa.

Figura 11. **Configuración diferencial del THS4503**

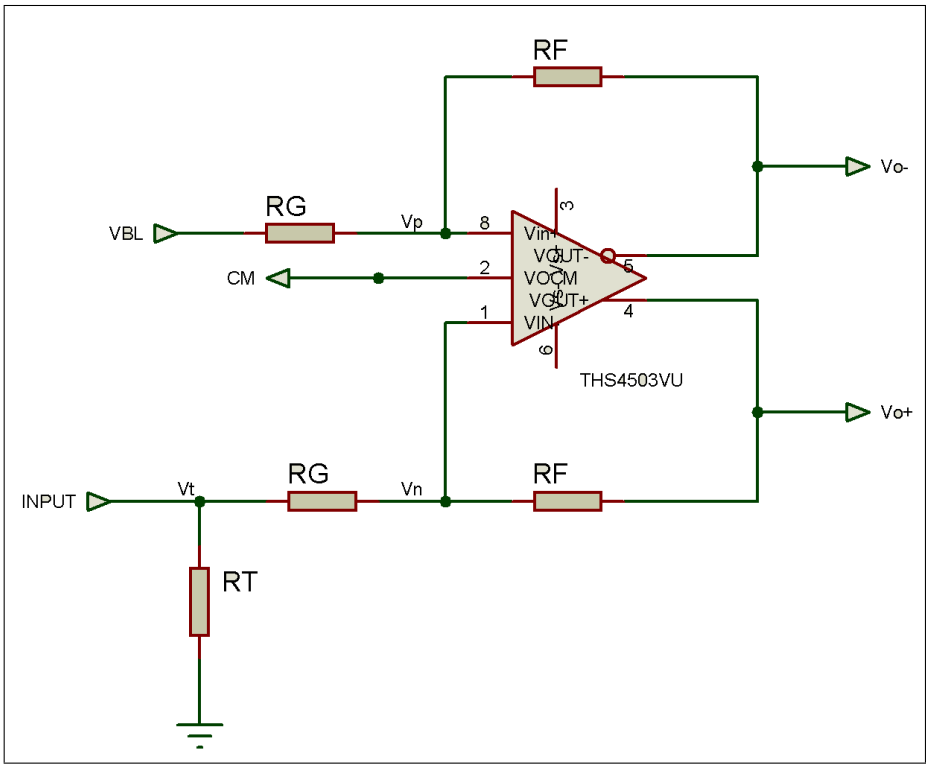


Fuente: *Wideband, Low-distortion Fully differential Amplifier*. p. 2.

La figura 11 muestra la configuración diferencial del amplificador. Sin embargo se le debe agregar una resistencia de acople R_T, la cual facilitará el acople de impedancia, así como se observa en la figura 12.

Del circuito que se muestra en la figura 12 se obtienen las ecuaciones 3.3. a la 3.7. donde; Z_{in} es la impedancia de entrada, I_{in} es la corriente de entrada, V_{ocm} es el voltaje de salida de modo común.

Figura 12. **Configuración del THS4503 para la tarjeta de adquisición (esquema simplificado)**



Fuente: elaboración propia, con el programa Proteus.

$$Z_{in} = \frac{V_T}{I_{in}} \tag{3.3.}$$

$$V_{ocm} = \frac{V_p + V_n}{2} \tag{3.4.}$$

$$\frac{V_{BL} - V_P}{R_g} = \frac{V_P - V_{O-}}{RF} \tag{3.5.}$$

$$\frac{V_n - V_{O+}}{R_f} = \frac{V_T - V_n}{R_G} \tag{3.6.}$$

$$I_{in} = \frac{V_T}{R_T} + \frac{V_T - V_n}{R_G} \tag{3.7.}$$

Tabla III. **Resistencias teóricas y comerciales para conseguir una impedancia de entrada de 50Ω**

Resistencia	Valor teórico (Ω)	Valor comercial(Ω)
R_f	6800	6800
R_g	2956	3000
R_T	50,5571	51

Fuente: elaboración propia.

Del juego de ecuaciones anterior se pueden deducir dos ecuaciones importantes:

$$H = -\frac{R_f}{2R_g} \tag{3.8.}$$

$$Z_{in} = \frac{2R_T R_g(R_f + R_g)}{2(R_g + R_t)(R_f + R_g) - R_f R_T} \tag{3.9.}$$

Donde H es la función de transferencia del circuito y Z_{in} es la impedancia de entrada.

Para un voltaje de entrada máximo de $-1,0V.$ y sabiendo que el voltaje de salida debe ser $1,15V.$ $(2,3V/2)$ y con la ecuación 3.8. se determina la relación:

$$R_f = 2,3R_g \tag{3.10.}$$

Introduciendo la relación 3.10. con la ecuación 3.9. se obtiene:

$$Z_{in} = \frac{R_g * R_T}{R_g + 0,681515R_T}$$

Para una impedancia de entrada deseada de 50Ω las resistencias teóricas y reales se detallan en la tabla III

Para completar el análisis. Se puede demostrar con las ecuaciones 3.3. a la 3.7. se puede llegar a la siguiente relación:

$$(V_{o+} - V_{o-}) = (V_n - V_p)(1 + \frac{R_f}{R_g}) - \frac{R_f}{R_g}(V_T - V_{BL})$$

Partiendo de ahí se puede obtener:

$$V_{O+} = H * (V_T - V_{BL}) \tag{3.11.}$$

Sin embargo se observa qué V_T es el voltaje de la señal más el voltaje en DC que se quiere suprimir con el V_{BL}, por lo que se satisfacen las tres condiciones necesarias para el acople de la señal.

3.3.1. Conversor análogo digital ADS5500

Este integrado es la parte principal de cada canal, ya que es el que envía la comunicación directa a la unidad de control.

El ADS5500 es un ADC de Texas Instruments de alta velocidad de adquisición y gran resolución.

Sus características principales son:

- Alta frecuencia de muestreo (125 MSPS).
- Alta relación señal-ruido (71.2 dBFS a 100 Mhz).
- 14 bits de salida en paralelo con bit de *overdrive* y *clockout* de sincronización.
- Empaquetado TQFP-64 con Power PAD.
- Bajo consumo de potencia (578 mW).
- Mínima dependencia de hardware externo.

Tabla IV. **Comparación entre ADC's utilizados en la electrónica de LAGO**

Característica	Bariloche AD9203ARU	México AD9216	Guatemala ADS5500
Resolución (Bits)	10	10	14
Tasa de muestreo (MSPS)	40	105	125
Consumo (mW)	78	300	578
SNR (dB)	59,5	57,6	71,2
Entrada diferencial	No	Si	Si
Canales	1	2	1

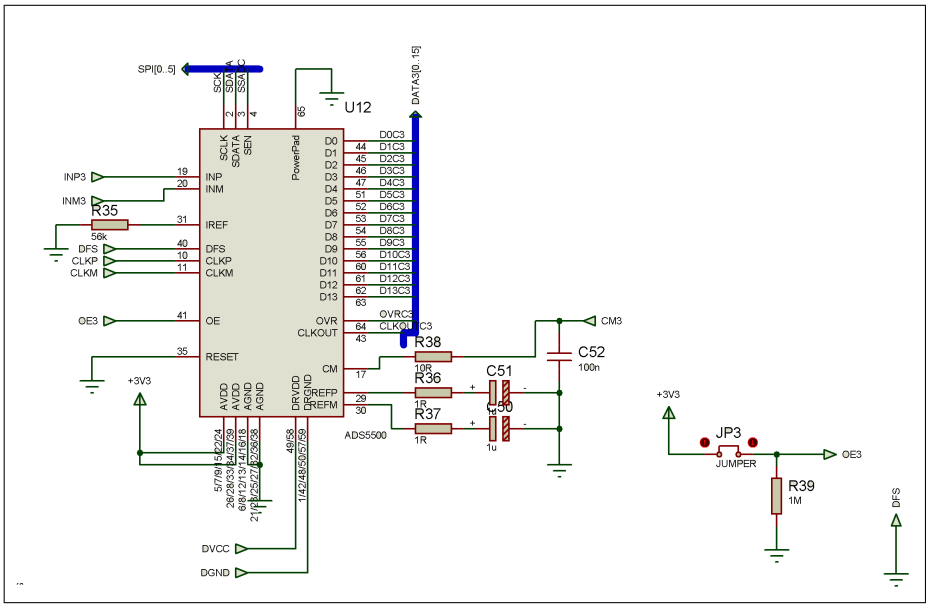
Fuente: elaboración propia.

En la tabla IV se muestra una comparación entre los ADCs utilizados en las diversas electrónicas implementadas. Se puede observar que, con excepción del consumo energético, todas las características del ADS5500 son superiores con respecto a los demás. La entrada diferencial permite tener un aislamiento del ruido externo lo cual mejora la calidad de la señal entrante.

En la imagen 13 se observan las conexiones a realizar en el ADS5500. El ADC necesita dos fuentes aisladas de voltaje, un voltaje analógico y uno digital, cada una con sus respectivas tierras.

El ADC necesita tres referencias para las salidas las cuales deben ser aisladas. $REFP$ Y $REFM$, que son aisladas por un capacitor de un microfaradio y una resistencia de 1 ohmio. La tercera referencia, $IREF$ es una salida de corriente necesaria para el funcionamiento y según la hoja de datos del ADS5500.

Figura 13. **Conexiones del ADS5500 en un canal**



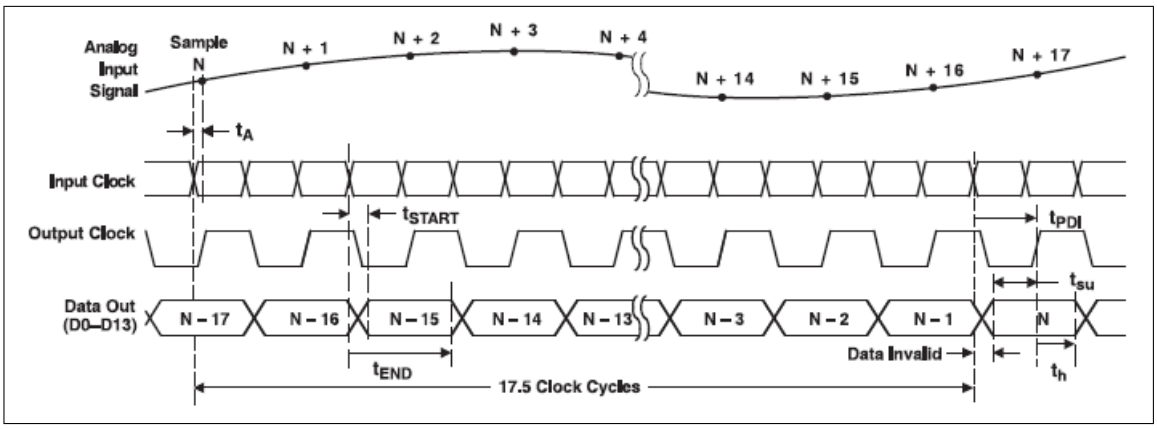
Fuente: elaboración propia, con el programa Proteus.

La salida es de 14 bits en paralelo con un pin *CLKOUT* de sincronización y un pin de *Overdrive* para una lectura fuera de escala. El formato binario de salida de datos *DFS* es escogido por el voltaje en el pin 40. El formato de elección es binario convencional *straight binary* con una validación de datos en el flanco positivo.

El ADS5500 tiene tres pines de entrada. Dos entradas diferenciales y una *CM* que acopla el voltaje en modo común proveniente del THS4503. El voltaje de entrada debe ser de 2,3 voltios entre las entradas diferenciales. En las entradas se recomienda colocar resistencias de 22Ω en el caso de un voltaje de entrada mayor o menor a los 2,3 voltios. Si el voltaje de entrada sale de la escala positiva, la salida del ADC es $0X3FFF$, en el caso de un *overdrive* negativo la salida es $0X0000$.

El ADS5500 realiza su conversión por medio de un *pipelining*, eso retrasa el procesamiento en 17 ciclos de reloj. Después de los primeros 17 ciclos realiza una captura en cada ciclo a la frecuencia del reloj de entrada. El proceso completo se muestra en la figura 14.

Figura 14. **Diagrama temporal de adquisición de datos en el ADS5500**

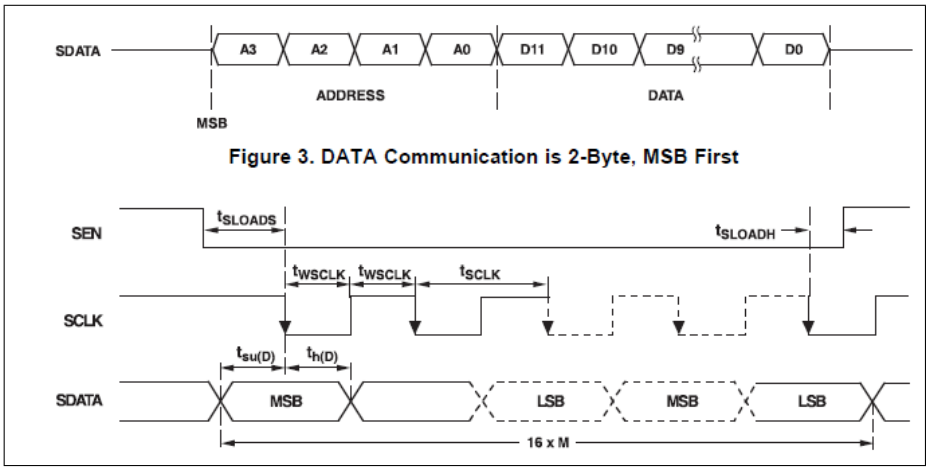


Fuente: hoja de datos: *14-Bit, 125Msps ANALOG-TO-DIGITAL CONVERTER*. p. 6.

3.3.1.1. Esquema de comunicación

Entre otras características, el ADS5500 tiene un control de funciones vía SPI Por medio de este bus de control, se puede programar el ADS5500 para entrar al modo de prueba y manejar remotamente el encendido, reset y apagado del ADC. El diagrama de control SPI se muestra en la figura 15.

Figura 15. **Diagrama de tiempo para el control SPI en el ADS5500**



Fuente: hoja de datos: *14-Bit, 125Msps ANALOG-TO-DIGITAL CONVERTER*. p. 8.

3.3.2. Desacople de las fuentes de poder

El desacople de las fuentes de poder es un aspecto crítico para el desempeño adecuado de circuitos de alta velocidad.

3.4. *Baseline control*

El *baseline control* tiene la tarea de nivelar el voltaje DC producido por el PMT en su condición de *standby*. Usualmente el valor DC del PMT oscila entre $\pm 1V$. si se recuerda la ecuación 3.11. se puede determinar que:

$$V_T = V_e + V_p$$

donde V_e es el voltaje cambiante en un evento y V_p es el voltaje en DC proveniente del PMT. Dado que la resta la realiza el THS4503 solo hace falta el V_{BL}, para eso se utiliza el DAC7614. Este es un DAC de 12 bits de resolución con operación bipolar y bajo consumo (20mW).

Ya que la principal variante del voltaje base del PMT es la temperatura y la electrónica no será expuesta a cambios bruscos de ella, el DAC no necesita alta velocidad de transición. La característica principal para la selección de este DAC es la operación bipolar, la cual simplifica en gran manera el barrido de voltaje en el V_{BL}, y el hecho de que tenga cuatro canales permite usar un solo integrado para todos los canales.

En la figura 44 se observa la configuración del DAC7614 para el *baseline control*, para aprovechar al máximo la resolución de los 12 bits se baja el voltaje de referencia a 1,1 voltios por medio de un arreglo de resistencias. El *reset* se configura que sea a media escala, esto indica que al momento de que el control reciba la señal de *reset* el voltaje de salida en los canales sea de 0 (*reset* a media escala).

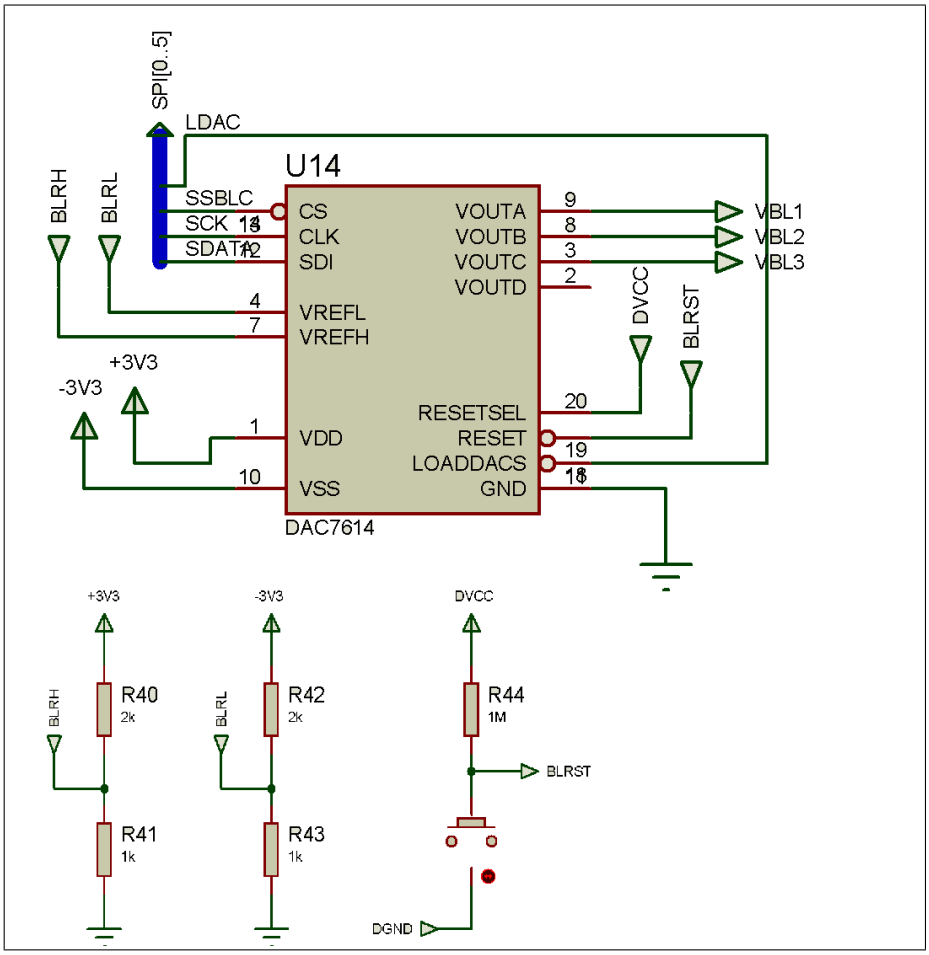
Por último, este DAC tiene una impedancia de salida alta, ya que tiene un *buffer* de salida. Esto evita tener que realizar un acople de impedancia al THS4503 reduciendo en gran manera el hardware necesario.

3.4.1. Esquema de comunicación

La comunicación con el DAC se realiza por un protocolo SPI de 16 bits. Los dos bits más significativos son utilizados para seleccionar la salida. Los dos siguientes no son usados (ver la hoja de datos), y los doce restantes son los del valor a convertir.

Sin embargo, el DAC7614 solamente cargará los valores cuando el pin $\overline{LOADDAC}$ cambie de flanco a bajo y luego regrese a un estado alto.

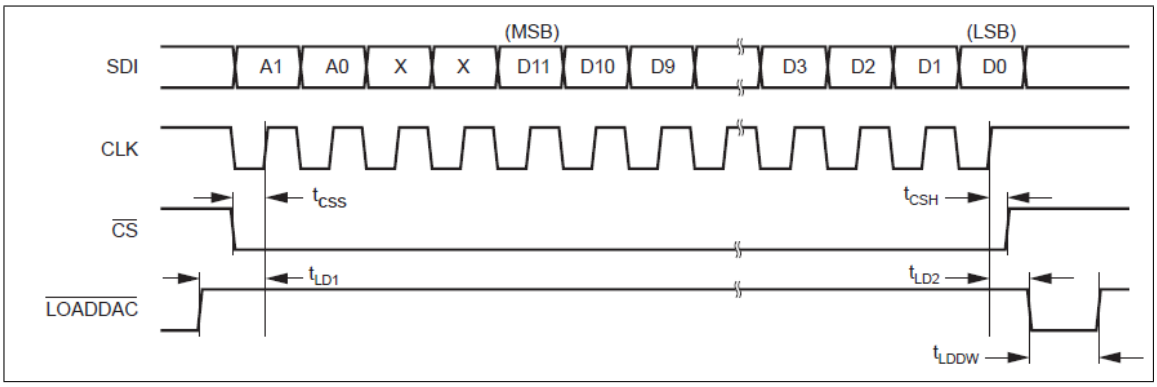
Figura 16. **Configuración del DAC7614 para el *baseline control***



Fuente: elaboración propia, con el programa Proteus.

El diagrama de tiempos se muestra en la figura 17 y la lógica de control se describe en la tabla V

Figura 17. **Diagrama de tiempos del DAC7614**



Fuente: hoja de datos: *Quad, Serial Input, 12-Bit, Voltage Output DIGITAL-TO-ANALOG CONVERTER DAC7614* p. 4.

Tabla V. **Lógica de control del DAC7614**

A1	A0	$\overline{LOADDAC}$	$\overline{RESET}$	DAC SELECCIONADO	ESTADO
L	L	L	H	A	Transparente
L	H	L	H	B	Transparente
H	L	L	H	C	Transparente
H	H	L	H	D	Transparente
X	X	H	H	NONE	Todos Asegurados
X	X	X	L	ALL	Reset

Fuente: BROWN, *Quad, Serial Input 12 Bit, Voltage Output*. p. 11.

Por ejemplo, si se quiere convertir el valor $3\,634$ ($0b111000110010$) a la salida C del DAC, el paquete enviado sería:

Tabla VI. **Paquete enviado para valor 3634 en decimal.**

$A1$	$A0$	X	X	$D11$	$D10$	$D9$	$D8$	$D7$	$D6$	$D5$	$D4$	$D3$	$D2$	$D1$	$D0$
1	0	X	X	1	1	1	0	0	0	1	1	0	0	1	0

Fuente: elaboración propia.

3.4.2. Codificación de datos de entrada

El formato de manipulación de datos del DAC7614 es de completo binario, es decir que se tienen 4 096 pasos (12 bits). La ecuación que domina el voltaje de salida se expresa en 3.12.

$$V_{OUT} = V_{REFL} + \frac{(V_{REFH} - V_{REFL}) * N}{4096} \tag{3.12.}$$

Donde: N es el código (en decimal) y V_{REFH} y V_{REFL} son los voltajes de referencia alto y bajo respectivamente.

Continuando con el ejemplo anterior, para un voltaje de referencia V_{REFH} y V_{REFL} de $1,1V.$ y $-1,1V.$ respectivamente; y un código recibido de $3\,634$. El DAC tendrá una salida de $V_{OUT} \approx 852mV.$

La ecuación anterior no expresa el *offset* en el voltaje de salida $V_{OUT} = 0$ el cual se denomina voltaje de escala cero. Dada la naturaleza bipolar del filtro, el voltaje de escala cero debe determinarse *in-situ*. Para los fines de la aplicación el efecto del *offset* no es significativo.

3.5. *Slow control*

La tarea del *slow control* es la de controlar la polarización del PMT. Se le llama control lento porque, comparado con el resto de la electrónica, esta parte del circuito es la menos dependiente del tiempo de respuesta.

Como se explicó anteriormente, el PMT utiliza una placa de polarización dependiente de voltaje. Cada cambio de $0,25V.$ en el voltaje de control representa un cambio de $100V.$ en la polarización del PMT. Para controlar el voltaje de control se utiliza el DAC de precisión TLV5614.

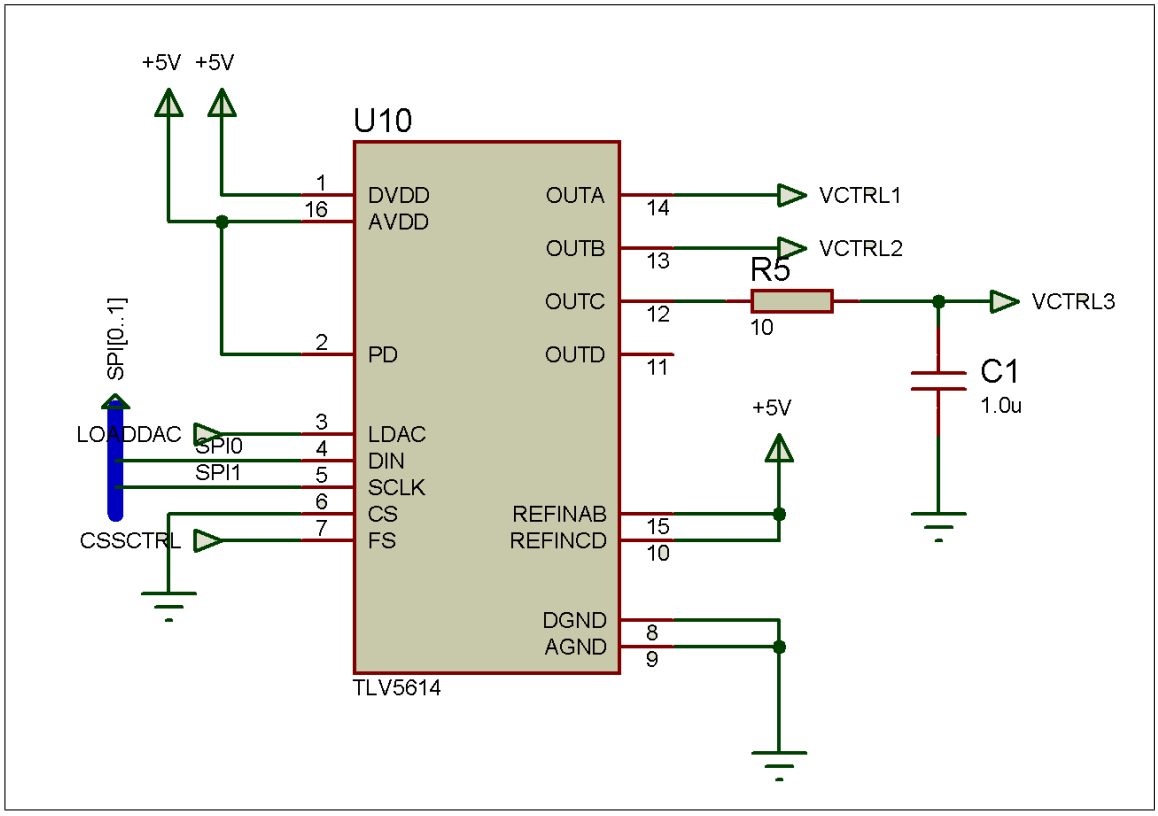
Este DAC fue seleccionado por su alta estabilidad, bajo consumo energético ($8mW.$ a $5V.$), cuatro salidas independientes, ninguna necesidad de componentes externos, bajo precio e interfaz de comunicación SPI compatible con el resto de la electrónica.

El voltaje de salida depende de la ecuación 3.13. donde REF es el voltaje de referencia, N es el código a convertir.

$$V_O = 2 * REF * \frac{N}{4096}[V.] \tag{3.13.}$$

El TLV5614 utiliza un esquema de conversión de duplicado de voltaje; para que esto se logre, el voltaje de referencia debe ser menor a la mitad del voltaje de alimentación para evitar la saturación en la salida. En la figura 18 se observa que el voltaje de referencia $REFINAB$ Y $REFINCD$ están conectados a 5 voltios el cual es también el voltaje de alimentación. Esta discrepancia en el diseño con la recomendación de la hoja de datos no es accidental.

Figura 18. **Configuración del TLV5614 para el *slow control***



Fuente: elaboración propia, con el programa Proteus.

Para las condiciones planteadas en el circuito, la ecuación 3.13. se puede reescribir de la siguiente manera:

$$V_O = 10 * \frac{N}{4\,096}[V.]$$

Para un $N = 4\,095$ (la entrada máxima), $V_O \approx 10V$ dado que la fuente de alimentación es de 5 voltios, esto no se puede cumplir y el circuito entra en saturación con una salida de 5 voltios. A simple vista parece un error de diseño, sin embargo, se estudia un caso específico. Para un voltaje de salida $V_O = 0,25V.$ el cual es el mínimo voltaje necesario para polarizar a 100 Voltios el PMT. Despejando para N se observa:

$$N = \frac{0,25 * 4\,096}{10} = 102,4 \approx 103$$

Ya que la ecuación 3.13. es lineal, cada incremento de 0,25 voltios en el voltaje de salida será reflejado en un incremento de aproximadamente 103 unidades en el código del DAC, que a su vez significa un aumento de 100 voltios en el PMT. En términos de programación, este código simplifica en gran manera el control del voltaje de polarización del PMT. Por ejemplo, si se quiere polarizar el PMT a 700 voltios, el código a enviar sería 721, para 1 200 voltios el código sería 1 236 y así sucesivamente.

El esquema de comunicación del TLV6514 es SPI a 3 bits, para actualizar la salida del DAC, al igual que el DAC7614, utiliza un bit LDAC.

3.5.1. Corrección de swicheo por carga

Experimentalmente se observó que el TLV5614 tiene problemas de estabilidad al someter la salida a una carga capacitiva alta (e.g. un cable largo), el efecto de dicha carga sobre la salida del integrado produce un swicheo a una frecuencia aproximada de 500 Khz (la frecuencia de swicheo del integrado en modo *low speed*)alrededor del valor nominal de la salida de voltaje. Para corregir este swicheo y, a la vez, evitar el ruido eléctrico, se utiliza un filtro pasa bajo pasivo con una frecuencia de corte de 50 Hz. el cual garantiza la estabilidad en voltaje de control del PMT.

3.6. Fuente de voltaje

La electrónica LAGO necesita cuatro niveles de voltaje:

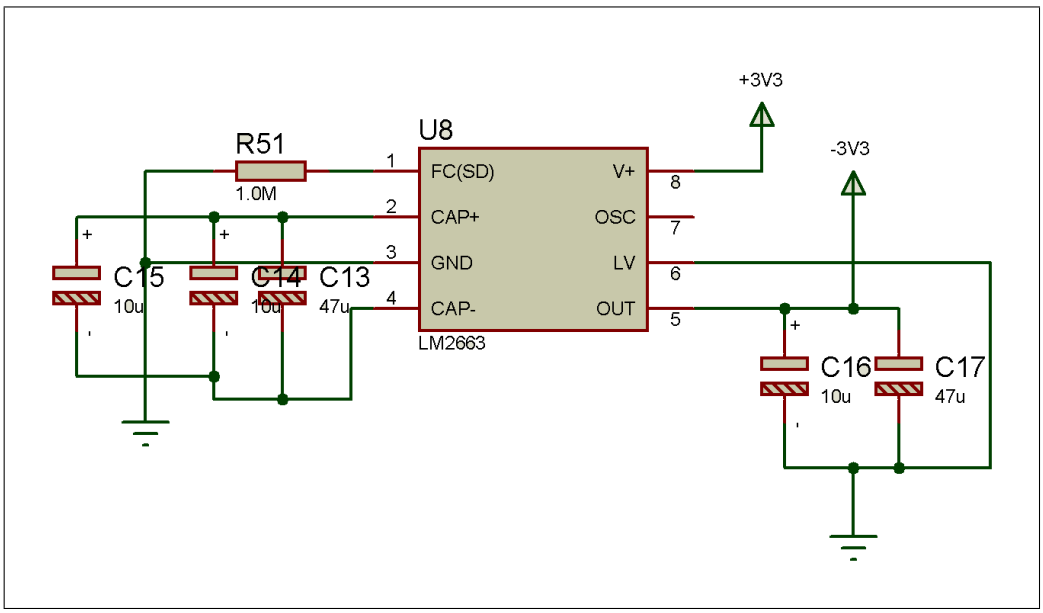
- +12,0V
- +5,0V
- +3,3V
- -3,3V

Además de la tierra, para suplir estos niveles de voltaje se detalla el circuito a utilizar.

Una de las ventajas de la tarjeta NEXYS 2, es que tiene un regulador de voltaje de 3.3 V. y soporta una entrada hasta 12.0 V. La NEXYS 3, tiene un regulador de voltaje de 3.3 V. y soporta un voltaje de entrada de 5.0 V. Basado en estas especificaciones, se tomó la determinación de alimentar las tarjetas NEXYS (2 o 3) con un voltaje de +5.0 V.

Para obtener los voltajes de +12.0V. y -3.3V. se utilizó los integrados LM2663 Y MAX662A en la configuración que se observa en las figuras 19 y 20.

Figura 19. **Configuración del inversor de voltaje LM2663**



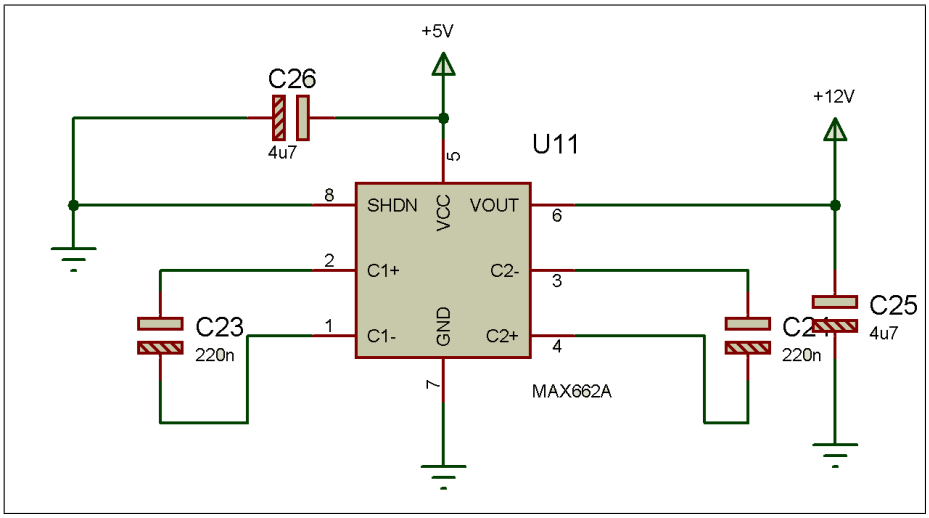
Fuente: elaboración propia, con el programa Proteus.

3.6.1. Inversor de voltaje de salida a -3,3V

El LM2663 es un inversor de voltaje por switcheo, la frecuencia de switcheo es de 20khz a 150khz. La estabilidad del inversor depende de la resistencia equivalente en serie (ESR por sus siglas en inglés) de los capacitores C13 y C17. El voltaje de rizado se determina por la siguiente ecuación:

$$V_{ripple} = \frac{I_L}{f_{osc} * C17} + 2 * I_L * ESR_{C17} \qquad (3.14.)$$

Figura 20. **Configuración del elevador de voltaje MAX662A**



Fuente: elaboración propia, con el programa Proteus.

Los capacitores de tantalio presentan un ESR, bajo lo cual lo hacen ideales para este diseño.

3.6.2. Elevación de voltaje de salida de 12,0V

Para elevar el voltaje de salida a 12,0 V, necesarios para polarizar el PMT se utiliza el integrado MAX662A. El voltaje es elevado por swicheo. Al igual que con el LM2663, el MAX662A produce un rizado dependiente del ESR de los capacitores en el circuito, dado que la corriente para polarizar el PMT es muy baja (en el orden de los microamperios), como se observa en la ecuación 3.14. el voltaje de rizado es mínimo. La configuración del circuito se observa en la figura 20.

3.7. Otras consideraciones

El circuito debe tener alta inmunidad al ruido. Para alcanzar esto, en el circuito se incluyen varios capacitores entre las salidas de voltaje (tanto positivas como negativas) y tierra, para disminuir el efecto de rizado.

Además de esto se colocan inductancias en serie entre las tierras digitales y analógicas para aislar los efectos de una sobre la otra. Así el switcheo en los conmutadores de voltaje no afectara las entradas digitales y las altas frecuencias de transmisión de datos, no afectara las salidas analógicas.

4. IMPLEMENTACIÓN DEL PROTOTIPO

En el siguiente capítulo se presenta la implementación del prototipo en placa del circuito impreso (PCB por sus siglas en inglés). Dada la naturaleza del circuito, la PCB debe ser diseñada tomando en consideración los efectos del *crosstalking*, el grosor longitud de las pistas de cobre. Además se presenta la depuración del sistema y las técnicas usadas para la corrección de errores.

4.1. Detección y corrección de errores

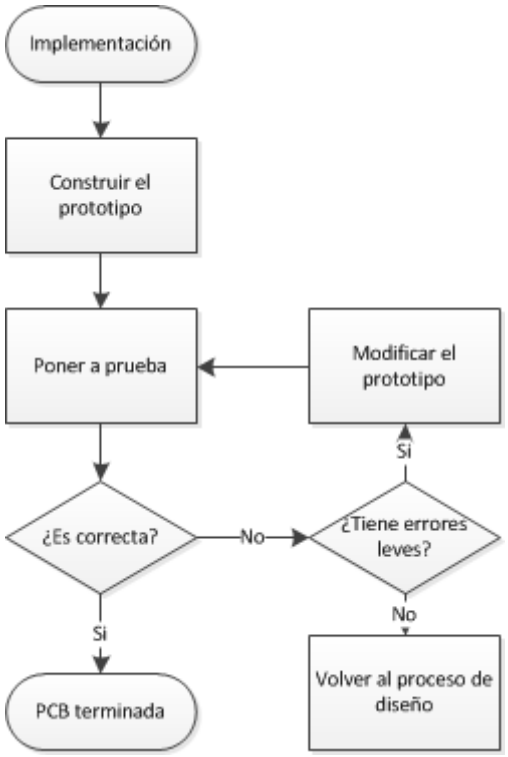
Cuando se diseña un circuito, la probabilidad de que el diseño funcione perfectamente es muy baja, casi siempre existen errores de diseño que deben ser corregidos antes de llegar a una electrónica definitiva.

En la figura 10 se expone el proceso utilizado para el diseño del circuito, sin embargo el diseño del PCB usualmente presenta ciertas complicaciones que no se pueden predecir al momento de diseñar un circuito en papel. Para el desarrollo del PCB se puede utilizar el proceso expuesto en la figura 21.

4.1.1. VHDL de prueba para LAGO-GT v 0.1

Con el propósito de probar la tarjeta de adquisición de datos, es necesario diseñar una plataforma de pruebas, la tarjeta de adquisición NEXYS 3 realiza. Esta tarea por medio de un *firmware* que permita probar el funcionamiento del sistema. Las características de dicho *firmware* deben permitir:

Figura 21. **Proceso para el desarrollo del PCB**



Fuente: BROWN s. & VRANESIC Z. *Digital Logic with VHDL Design*. p. 9.

- Probar el correcto funcionamiento del ADC así como encenderlo y apagarlo via remota.
- Probar el voltaje de salida del *slow-control*.
- Probar el voltaje de salida del *baseline control*.
- Enviar cada señal de prueba de manera independiente a cada componente SPI, para poder realizar mediciones dentro de la placa.

La descripción del lenguaje VHDL, así como el funcionamiento interno del programa, van mas allá de la exposición de este documento, el código fuente del *firmware* así como el de cada componente utilizado en VHDL se detallan en los anexos.

Cabe resaltar que este *firmware* de prueba no corresponde en ningún momento el *firmware* de adquisición de datos.

4.1.1.1. Funcionamiento del *firmware*

Una vez cargado el firmware en la FPGA, su funcionamiento es bastante simple:

La interfaz se controla por los cinco botones y los 8 switches, la retroalimentación visual es manejada por cuatro *display* de siete segmentos. Las salidas de control se manejan por los PMODS A,B,C de la NEXYS 3 según la configuración de pines que se observa en el capítulo anterior.

Al encender la NEXYS 3 se visualiza un 1234. Para empezar a utilizar el programa. Presione el botón central para acceder el menú. Las opciones del menú son las siguientes:

- ADC: señal de 16 bits para control del ADC
- SCL: señal de 16 bits a enviar al *slow-control*
- BCL: señal de 16 bits a enviar al *baseline control*

Se puede seleccionar la opción visualizada por medio de los botones arriba y abajo. Para ingresar a la opción deseada, presionar el botón central.

Si todos los switches se encuentran en bajo, la señal a visualizar será 0000 lo cual corresponde al valor hexadecimal a enviar. Para cambiar dicho valor, se manipulan los switches a modo de colocar en el display el byte menos significativo, con la flecha izquierda se puede subir al byte más significativo, el cual cargará el mismo valor desplegado en los switches, manipular los switches para cambiar el byte más significativo, en cualquier momento se puede volver utilizando el botón derecho, con el botón de arriba se sale de la configuración.

Para enviar el valor visualizado, presionar el botón central. El valor será enviado vía SPI a la unidad seleccionada por medio del PMOD C.

4.1.2. Construcción del prototipo

Dada la naturaleza modular del circuito (ver figura 9), cada bloque fue construido por separado para luego unificarlo en un solo canal.

El primer prototipo fue desarrollado en protoboard fabricando *break-out-boards* para los componentes de montaje superficial. En la tabla VII se detallan las pruebas realizadas de cada sub bloque del circuito y sus resultados.

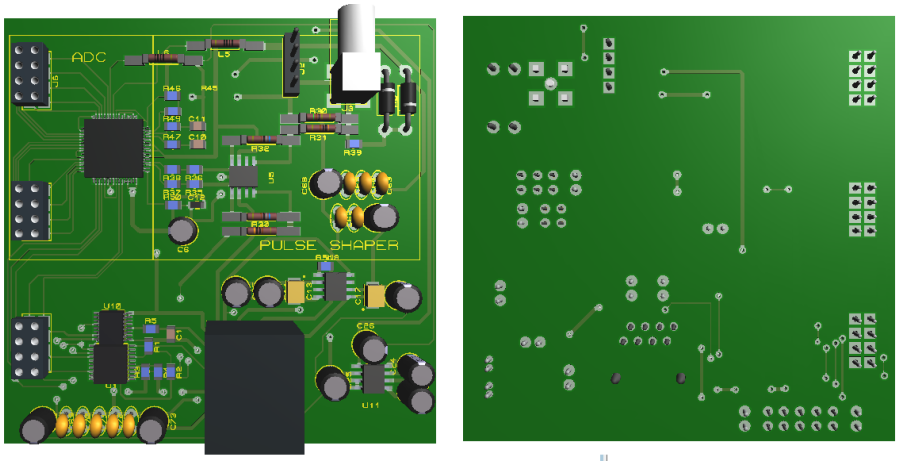
Tabla VII. ***Break-out-boards*** **fabricados y pruebas realizadas.**

Bloque	Pruebas realizadas	Resultados	Observaciones
Inversor de voltaje -3,3V.	$V_{out} = -3,3V.$	Voltaje de rizado a 150 Khz.	Hay rizado debido a la capacitancia propia de las placas del protoboard que aumentan la ESR de los capacitores de Tantalio.
Elevador de voltaje a 12,0V.	$V_{out} = 12,0V.$	Satisfactorio	
slow-control	Impedancia de entrada salida diferencial	Impedancia de entrada de 51Ω, salida diferencial correcta.	Consume mucha corriente provocando una caída en la entrada de -3,3V.
Base Line Control	V_{out} *Rail to rail*	Satisfactorio	
ADC	$CLKOUT$ Salidas en alta, baja, alternantes, SPI	Satisfactorio	Existe *crosstalking* utilizando cables, debe solucionarse en el PCB.

Fuente: elaboración propia.

Para corregir los errores observados en el protoboard, se trasladó el diseño a una placa de circuito impreso, PCB, el diseño del primer prototipo en PCB se observa en la figura 22 y la implementación del mismo en la 23, el cual fue llamado LAGO-GT v 0.1.

Figura 22. **Diseño del prototipo 1 en PCB**



Fuente: elaboración propia, con el programa Proteus.

4.1.3. Pruebas realizadas

La placa de adquisición fue sometida a una señal de prueba proveniente del PMT emulada por un led ultravioleta por un tiempo de 25 nanosegundos. Se evaluó cada especificación planteada en la tabla II, así como niveles de ruido y detección de falsos positivos. Se calificó utilizando los siguientes criterios:

- Satisfactorio (S): los resultados de la medición concuerdan con los parámetros especificados.
- Errores leves (L): los resultados no concuerdan con los parámetros especificados, sin embargo, son fácilmente corregibles dentro del prototipo por lo que no suponen inconveniente mayor.
- Errores graves (G): los resultados no concuerdan con los parámetros y se necesita un nuevo análisis y diseño.

Figura 23. **Implementación del prototipo núm. 1 en PCB**

Fuente: elaboración propia, con el programa Proteus.

La gravedad del ruido de swicheo provocado por los inversores de voltaje reside en su naturaleza parásita, el ruido se propaga no solo en la salida del inversor sino a través de la tierra infectando cada componente. Se cataloga como error grave dado que no se puede corregir directamente en la placa prototipo.

Tabla VIII. **Pruebas realizadas al prototipo LAGO-GT V0.1**

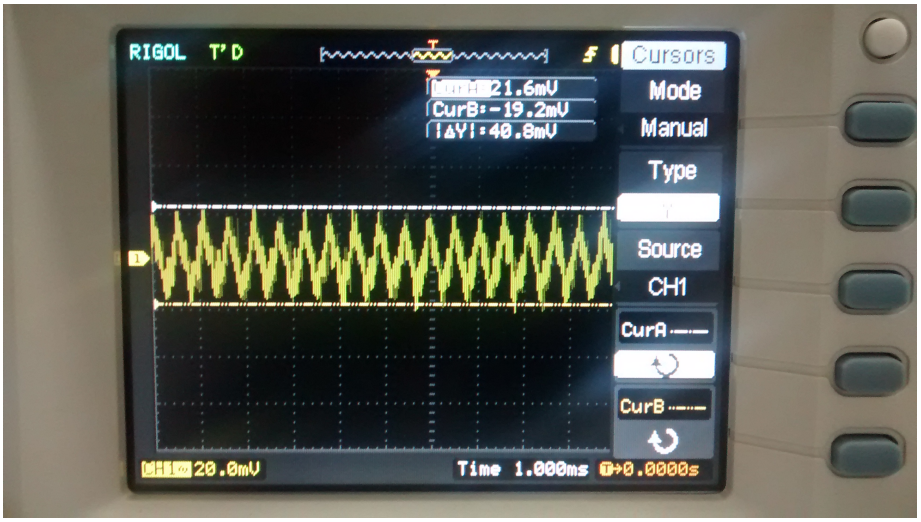
Descripción de la prueba	Metodología	Res	Observaciones
Encendidio, apagado y medición de las salidas provenientes del ADC.	Se enviaron las señales de prueba vía SPI y midió con el osciloscopio la respuesta a ellas.	L	La prueba es satisfactoria en señales fijas (todas las salidas en alto, todas en bajo) sin embargo con señales alternantes se observa *crosstalk*.
Voltaje de salida del *slow-control* sin carga	Barrido de voltaje enviando señales vía SPI.	S	
Voltaje de salida del *slow-control* con carga	Barrido de voltaje enviando señales vía SPI	L	Existe ruido inducido por la longitud del cable. Se solucionó con un filtro pasa bajo con $f_c = 50Hz$.
Voltaje de salida *baseline control*	Barrido de voltaje enviando señales vía SPI	S	
Voltaje de alimentación	Se midieron los voltajes de $+3,3V$, $-3,3V$. $+12,0V$, $+5,0V$.	G	Ruido de swicheo parasito.

Fuente: elaboración propia.

4.1.4. Corrección de errores

Los errores leves fueron reparados cambiando levemente el diseño inicial, en la figura 24 se observa el ruido en la salida del *slow-control* al conectarse el cable RJ-45. En la figura se observa el valor AC del ruido únicamente. El ruido oscila en una frecuencia de 2 KHz, con una amplitud de 40,8 mV p-p, ya que el voltaje de polarización del PMT cambia cada 25 mV se cataloga el rizado como error a corregirse.

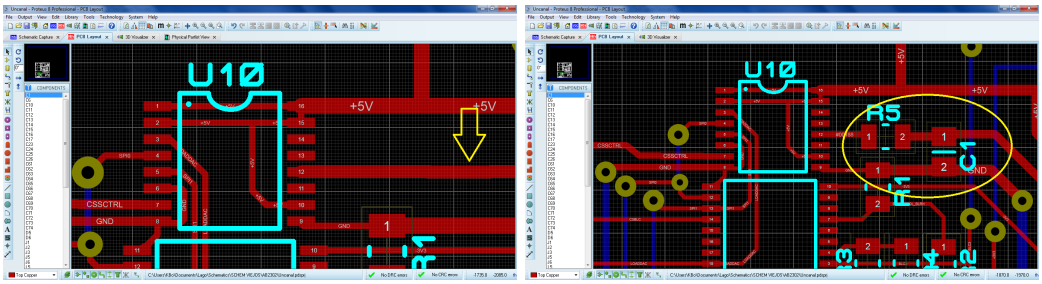
Figura 24. **Ruido en la salida del *slow-control* producido por la capacitancia del cable**



Fuente: elaboración propia.

La corrección de este error es simple, ya que basta agregar un filtro paso bajo pasivo con frecuencia de corte de aproximadamente 50 Hz logrado con una resistencia de 10 Ω y un capacitor de 1 μF. En la figura 25 se muestra el agregado al PCB del filtro.

Figura 25. **Agregado del filtro en la PCB prototipo**



Fuente: elaboración propia, con el programa Proteus.

El error por *crosstalk* es considerado leve, ya que la corrección del mismo depende únicamente del diseño del *layout* en el PCB y no de la electrónica de adquisición; ya que no se espera que el prototipo sea el encargado de la adquisición las correcciones se reservan a la fabricación del próximo prototipo.

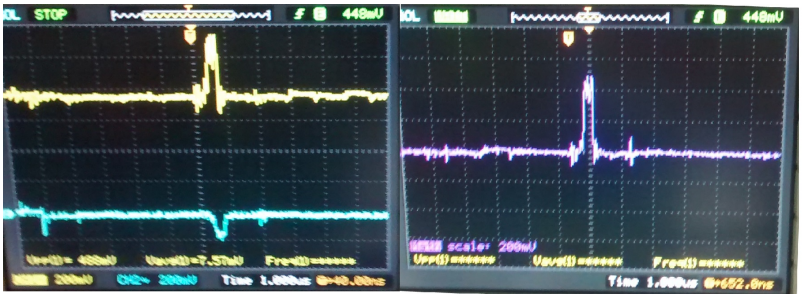
4.2. Errores graves

El ruido de swicheo provocado por los inversores de voltaje se cataloga como error grave porque se extiende por todo el circuito y la corrección del mismo depende tanto del diseño del PCB como del diseño del circuito.

4.2.1. Determinación del error

En la figura 26 se observa la salida de voltaje proveniente del amplificador diferencial, las lineas en amarillo y celeste expresan las salidas inversoras y no inversoras y la morada muestra la señal diferencial completa (como debería verla el ADC), en la gráfica de la derecha se observa un pico de voltaje a la derecha de la señal completa. Inicialmente se interpretó como un error aleatorio, sin embargo, al conectar la señal de entrada a tierra y medir (27) se observa que cada pico se repite a una frecuencia de 179 Khz con una amplitud de 39,2 mV. Esta señal se repite en cada una de las señales de voltaje así como en las líneas de tierra.

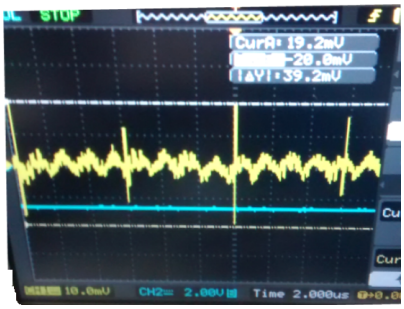
Figura 26. **Señal diferencial saliente del THS4503**



Fuente: elaboración propia.

Apagando el ADC, el *slow-control* y el *baseline control*, únicamente quedaron los inversores de voltaje y el THS4503, se procedió a remover el amplificador diferencial y cada uno de los inversores y se determinó que la fuente de ruido era el LM2663. Este ruido de swicheo entra al THS4503 y es amplificado provocando un error en la detección disparando la señal de adquisición dando como resultado falsos positivos.

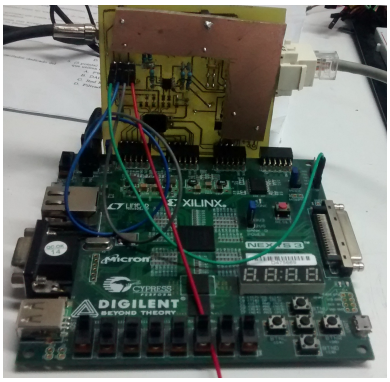
Figura 27. **Ruido de *swicheo* parásito**



Fuente: elaboración propia.

Para corregir este error se intentó aislar la señal de ruido utilizando inductancias de entrada y salida, así como agregar un panel de tierra virtual para mejorar el retorno de las cargas. El prototipo con las correcciones se observa en la figura 28 y en la 29 se muestra la salida diferencial, a pesar de que no es apreciable a simple vista, el ruido de swicheo persiste como se describe en la figura 29, hay un pico antes de la señal diferencial provocado por el swicheo.

Figura 28. **Prototipo con panel de tierra virtual**



Fuente: elaboración propia.

A pesar de la reducción del ruido de swicheo, el umbral de ruido es bastante alto, alrededor de 75mV, eso es aproximadamente 10 de los 14 bits desperdiciados en el umbral de ruido. A pesar de que se puede medir con éxito y detectar el error aumentando el valor del *Threshold* se pierde el propósito de aumentar el número de bits, proporcionando una solución poco elegante.

Con base en el proceso de diseño en la figura 21 ante la presencia de errores graves, se debe volver al proceso de diseño. Anotados ya los errores, se profundizará en cada uno de ellos con el fin de corregirlos en la versión definitiva del prototipo.

Figura 29. **Salida diferencial despues de las modificaciones**

Fuente: elaboración propia.

5. DISEÑO DEL SISTEMA DE ADQUISICIÓN DE DATOS PARA LAGO-GUATEMALA: CORRECCIÓN DE ERRORES Y DISEÑO FINAL

5.1. Consideraciones iniciales

En la sección anterior se encontraron errores graves en el prototipo LAGO-GTv 0.1. Estos obligan a un rediseño en aspectos claves del circuito, así como del PCB. Los errores críticos a corregir son los siguientes:

- Reducir al mínimo el ruido de swicheo producido por los inversores de voltaje.
- Corregir el crosstalk en la salida del ADC.

Con la finalidad de corregir estos errores, se estudió nuevamente el diseño inicial planteado en el capítulo 3. Se observó que muchos componente estaban trabajando en condiciones no optimas, esto, si bien no es un error, puede desestabilizar el sistema, por lo que se decidió rediseñar para garantizar la máxima estabilidad posible.

Las características a optimizar se detallan en la tabla IX. Entre otras características a cambiar, se determinó que las salidas de voltaje de $\pm 3.3V.$ no son necesarias en la polarización directa del PMT, por lo que se decidió quitarlas de la salida RJ-45. Al realizar este cambio, se aísla del PMT el voltaje negativo facilitando el manejo del ruido y disminuyendo la carga. Se deja la opción de utilizar una fuente externa de $\pm 3.3V.$ en la salida al RJ-45.

Tabla IX. **Características a optimizar en el nuevo prototipo**

		Condiciones	
Componente	Característica	Iniciales	Optimizados
THS4503	V_{Input}	$\pm 3,3V.$	$\pm 5,0V.$
DAC7614	V_{DD} y V_{SS}	$\pm 3,3V.$	$\pm 5,0V.$
TLV5614	V_{Ref}	$+5,0V$	$+2,5V$
TLV5614 y DAC7614	V_{SPI}	$+3,3V.$	$+5,0V.$

Fuente: elaboración propia.

Las correcciones realizadas no deben perjudicar las pruebas satisfactorias ya realizadas. La justificación de cada una de estas correcciones se detalla en las próximas secciónes.

5.2. Corrección del ruido de *swicheo*

El LM2663 es un inversor utilizado del tipo *Charge pump* o de bombeo. Como se observó en el capítulo 3, el voltaje de rizado (V_{ripple}) se define en la ecuación 3.14.

$$V_{ripple} = \frac{I_L}{f_{osc} * C} + 2 * I_L * ESR_C \tag{3.14.}$$

Donde I_L es la corriente de carga, f_{osc} es la frecuencia de oscilador del inversor de voltaje, C es el capacitor de carga y ESR es la resistencia estimada en serie del mismo.

Como se observa en la ecuación, siempre que exista carga habrá voltaje de rizado. Ya que no se puede eliminar, se puede aislar. Según la nota técnica de Maxim *Maxim charge pump ripple* el ruido de rizado se puede aislar por tres métodos:

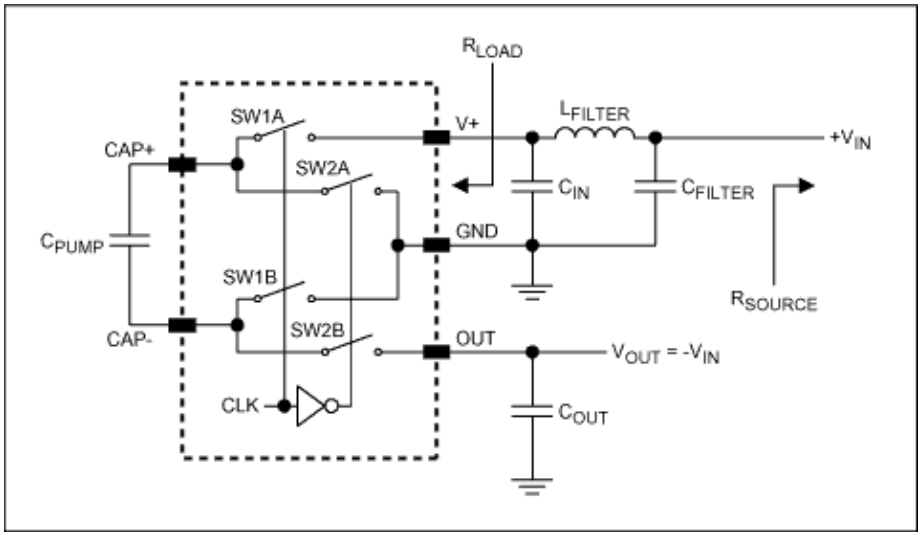
- Filtro LC
- Regulador LDO
- Filtro RC

En la figura 30 se observa la configuración del filtro LC para evitar la propagación del ruido en el resto del circuito. La frecuencia de corte es determinada con la ecuación 5.1. en donde $C_{Filter} = C_{In}$ y L_{Filter} es una inductancia de ferrita.

$$F_{-3dB} = \frac{1}{2\pi\sqrt{L_{Filter}C_{Filter}}} \tag{5.1.}$$

Se realizó una prueba utilizando un *Break out board* del LM2663 con y sin filtro de ferrita. En la figura 31 se observa el ruido de swicheo sin el filtro LC, la configuración del osciloscopio es a una escala de voltaje a 20 milivoltios por división y una escala temporal de 8 microsegundos. El nivel de voltaje pico a pico de oscilación es de 100 milivoltios. En la figura 32 se observa el resultado del mismo circuito inversor agregando un filtro LC la señal resultante tiene una amplitud pico a pico de 38 milivoltios significando en una atenuación de -8 dB.

Figura 30. **Filtro LC para reducción del ruido de rizado**

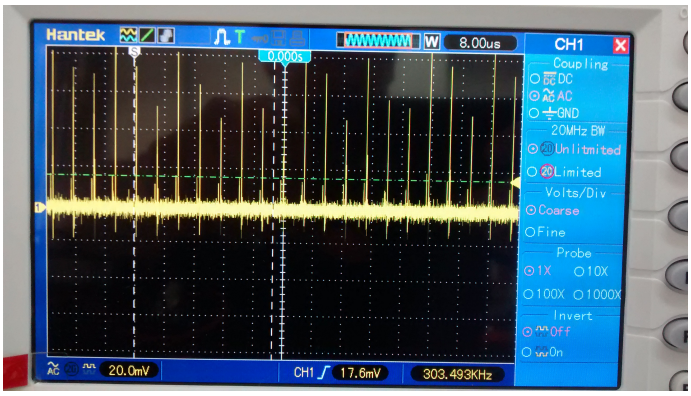


Fuente: MAXIM IC. *Simple Methods Reduce Input Ripple for All Charge Pumps*. p. 1.

La ecuación 3.14. se aplica para cualquier inversor del tipo *Charge pump*, La atenuación del ruido de swicheo es significativa, sin embargo, otro factor importante es la frecuencia de swicheo, al momento de aumentar la frecuenciea de swicheo, disminuye el rizado. El LM2663 tiene una frecuencia de swicheo de 150Khz (179 KHz experimentalmente) mientras tanto, el MAX889SESA tiene una frecuencia de swicheo de 1 Mhz. Utilizando el mismo filtro LC y cambiando al MAX889ESA se logra atenuar el ruido de swicheo a 22 mV. (-13 dB), regulando la salida con un LDO (MAX1735) el rizado se atenua a 17 mV p-p (-15 dB), los resultados se observan en las figuras 33 y 33. Se nota que la escala temporal continua siendo de 8μs más la escala de voltaje es de 10 mV.

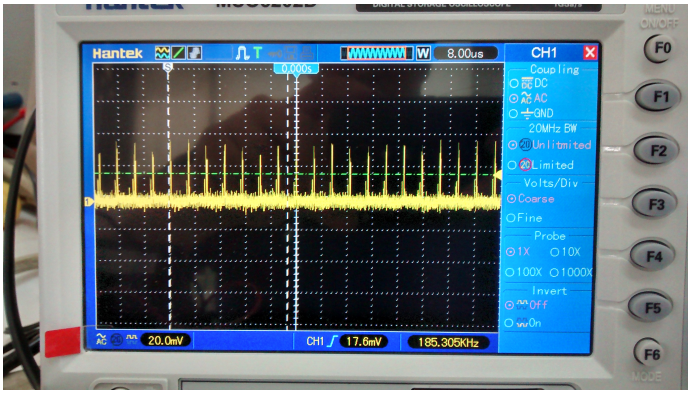
El circuito inversor de voltaje corregido se detalla en la figura 35 para un voltaje de salida de -5,0V.

Figura 31. **Voltaje de rizado sin filtro LC**



Fuente: elaboración propia.

Figura 32. **Voltaje de rizado con filtro LC**

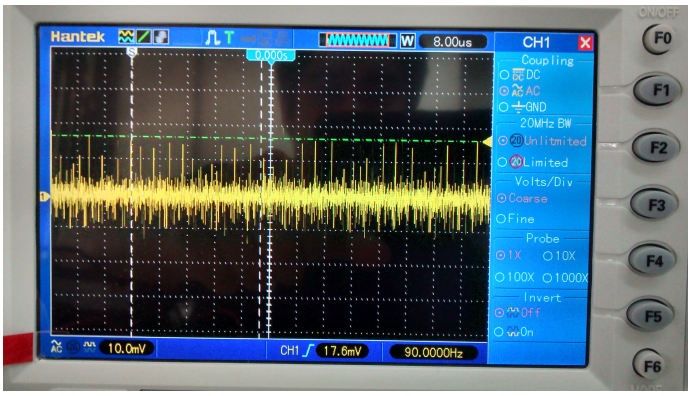


Fuente: elaboración propia.

5.2.1. Regulador de voltaje +5,0V

Otra fuente de ruido considerable, que no fue tomada en cuenta en el prototipo inicial, fue el de la fuente de alimentación.

Figura 33. **Voltaje de rizado utilizando el MAX889ESA sin LDO**



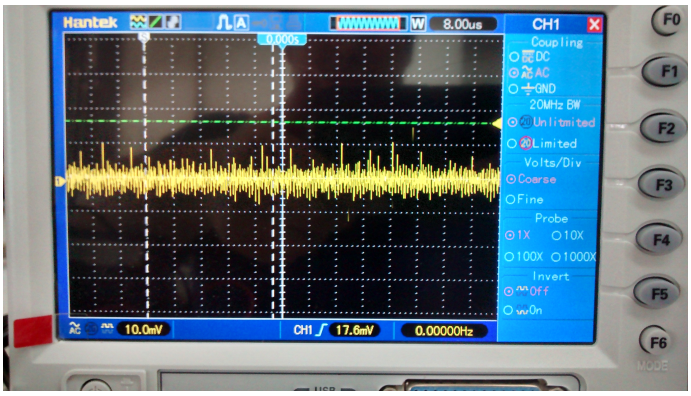
Fuente: elaboración propia.

Originalmente el voltaje de alimentación de +5.0V. provenía directamente de una fuente de voltaje externa, esta fuente alimentaba la NEXYS 3 y todos los componentes del *slow-control*.

El principal inconveniente de este diseño inicial radica en el hecho de que las fuentes de voltaje actuales utilizan reguladores de swicheo, lo cual puede ingresar ruido al sistema. Además, en caso de un corto circuito dentro de la placa o en la NEXYS 3, se corre el riesgo que ambas electrónicas sean dañadas.

Para corregir estos errores, se instaló un regulador LDO para separar el voltaje de alimentación de la NEXYS 3 con el resto de la electrónica. Para dicha tarea se empleó el integrado MAX1818EUT33, el cual es un regulador LDO con voltaje de salida de 500 mA programable.

Figura 34. **Voltaje de rizado utilizando el MAX889ESA con LDO**



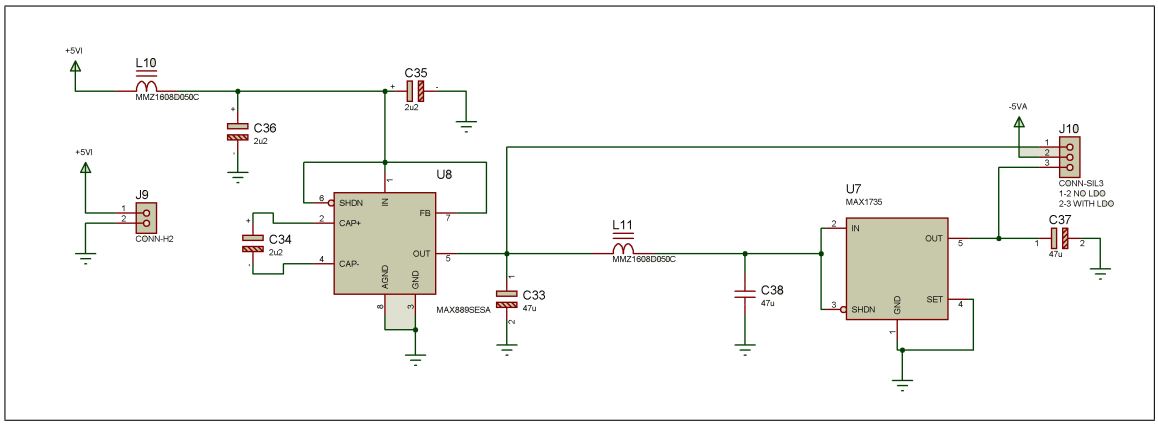
Fuente: elaboración propia.

La figura 36 muestra el esquemático del regulador de voltaje. En la salida se encuentra un *Jumper* (J11), el cual permite seleccionar el voltaje de alimentación a usar en los componentes, ya sean regulados o sin regular. Esta decisión depende de la fuente de voltaje a utilizar.

EL MAX1818EUT33 es un regulador programable en donde el voltaje de salida está regido por la ecuación

$$V_O = \left(\frac{R_A}{R_B} + 1\right) V_{Set} \tag{5.2.}$$

Figura 35. **Circuito inversor de voltaje para -5,0 V**



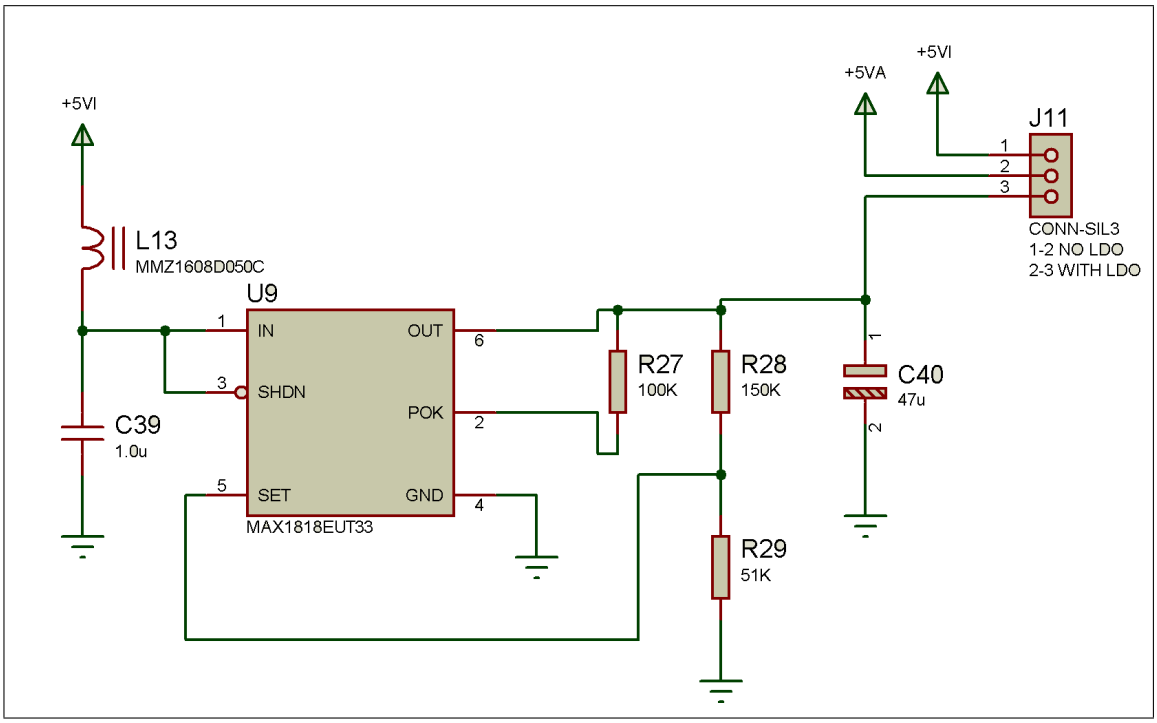
Fuente: elaboración propia, con el programa Proteus.

Donde $V_{Set} = 1,25V$ y para el diagrama en la figura 36 las resistencias $R_A = R_{28}$ y $R_B = R_{29}$ y V_O puede variar en el rango de 1,25 voltios a 5,0 voltios. Por defecto el MAX18182UT33 tiene un voltaje de salida de 3,3 voltios al colocar SET a tierra. Dado que todos los integrados utilizados en la digitalizadora pueden usar un voltaje de entrada de $+3,3V.$ o $+5,0V.$ se puede cambiar la configuración del regulador desoldando la resistencia 28 y cambiando por una de 0Ω la resistencia 29. Sin embargo, no se recomienda bajar el voltaje de alimentación con otro propósito además de pruebas del sistema.

5.2.2. Elevador de voltaje de +12,0V

Para elevar el voltaje de salida de +12,0 voltios se utiliza el MAX662A el circuito original se detalla en la figura 20, dado que es un circuito integrado de bombeo, tambien puede ingresar ruido al sistema, para evitar eso se utilizan ferritas y capacitores de desacople como se observa en la figura 38.

Figura 36. **Regulador de voltaje para salida de +5,0V**

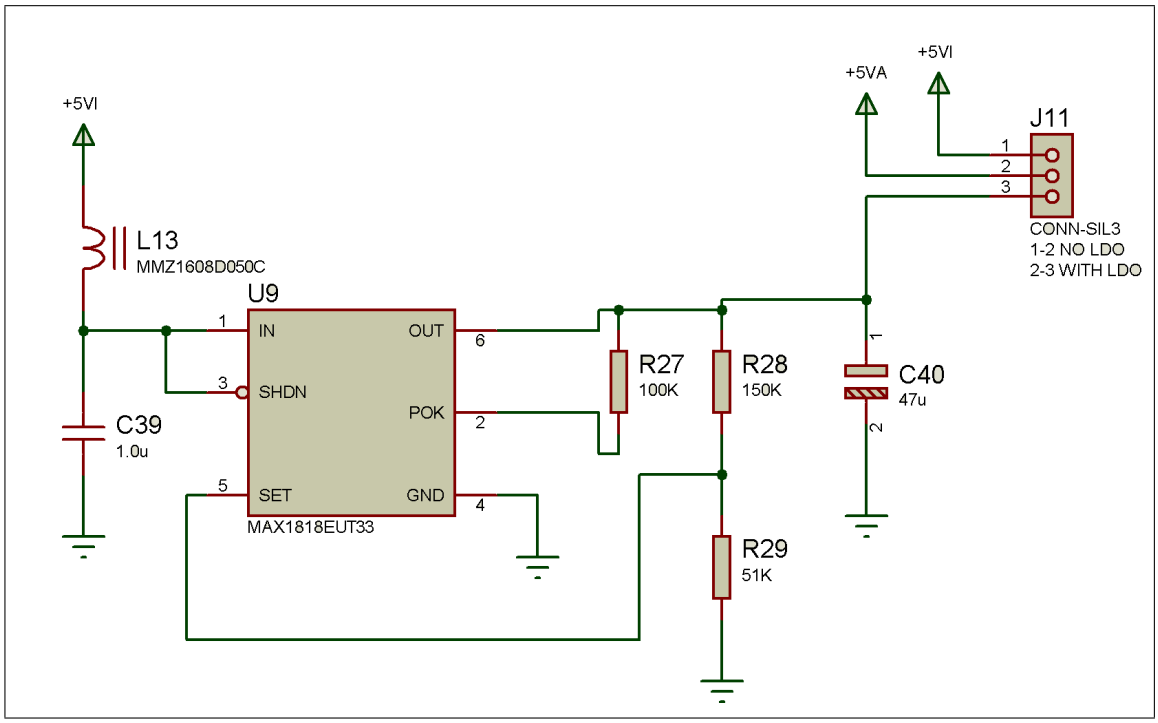


Fuente: elaboración propia, con el programa Proteus.

5.2.3. Capacitores de desacople

Los capacitores de desacople, entre los pines de voltaje y de tierra aseguran baja impedancia al ruido en AC. Para llegar a esta baja impedancia entre un rango de frecuencias específico es necesario un arreglo de distintos valores de capacitores.

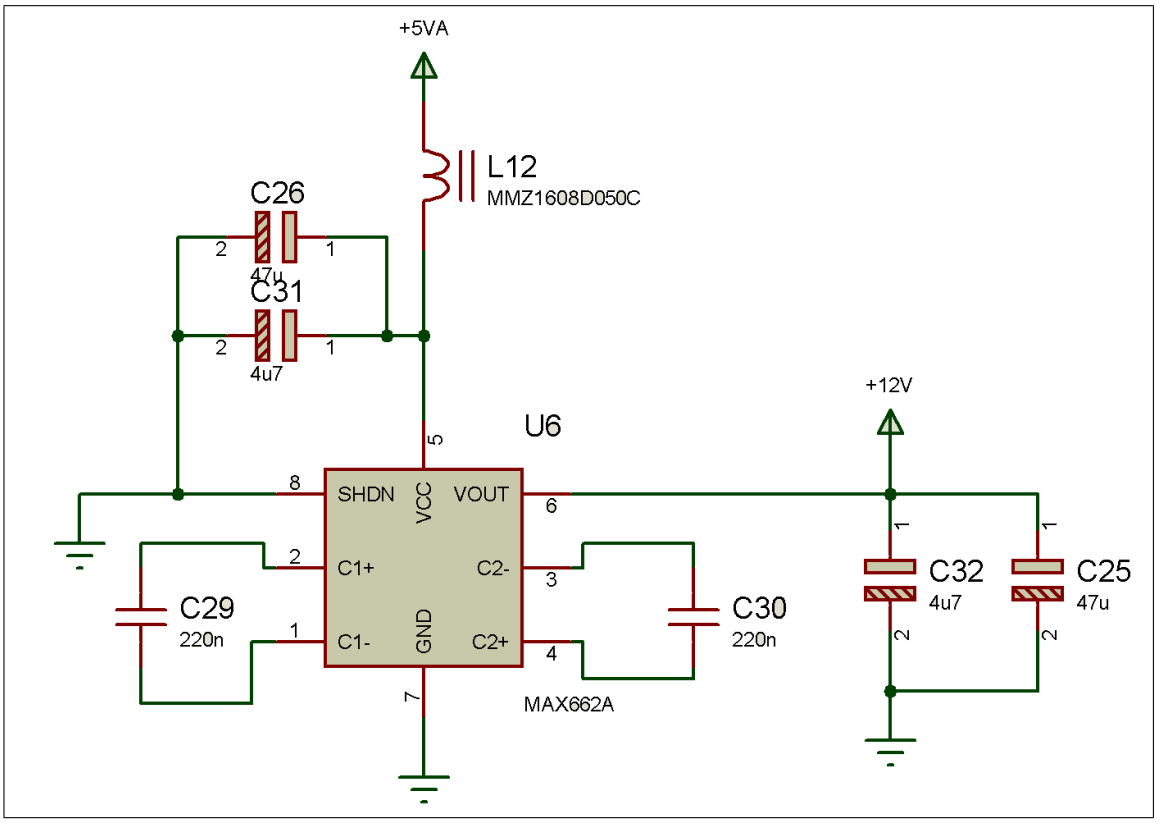
Figura 37. **Regulador de voltaje**



Fuente: elaboración propia, con el programa Proteus.

La figura 39 muestra la impedancia de diversos valores de capacitores en un rango de frecuencias variadas así como la respuesta en paralelo (línea azul), ya que no existen capacitores ideales, cada capacitor puede entrar en resonancia propia en donde la impedancia se dispara, por esta razón es importante colocar diversos valores de capacitancia en paralelo para eliminar la mayor cantidad de componentes de resonancia posibles.

Figura 38. **Elevador de voltaje MAX662, con desacople de ferrita**

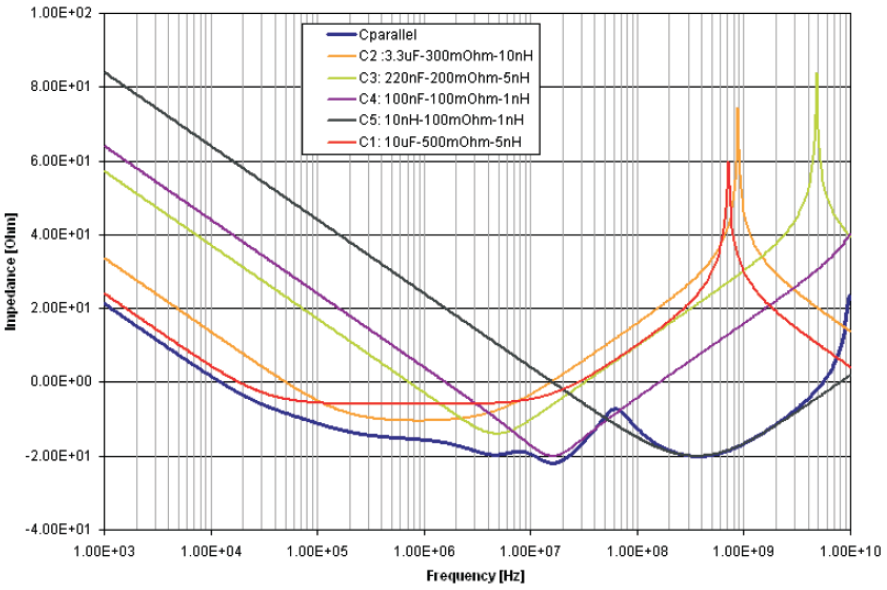


Fuente: elaboración propia, con el programa Proteus.

5.2.4. Otras consideraciones

Es muy importante separar las fuentes de voltaje a utilizar para señales analógicas como para digitales. Las señales análogas pueden transmitir ruido a las señales digitales. Para prevenir esto, basta con aislar las señales análogas con ferrtia de choque. Las ferritas utilizadas tienen impedancia de entrada de 0,5Ω en DC y 600Ω a 100 MHz. De ahora en adelante se determinarán como $xxVA$ para voltaje analógico y $xxVD$ para referirnos a voltaje digital, cada señal analógica será separada de las digitales, por lo menos por una ferrita.

Figura 39. **Impedancia de diversos capacitores en un rango amplio de frecuencias**



Fuente: WEILER A, PAKOSTA, A. *High-speed layout guidelines*. p. 13.

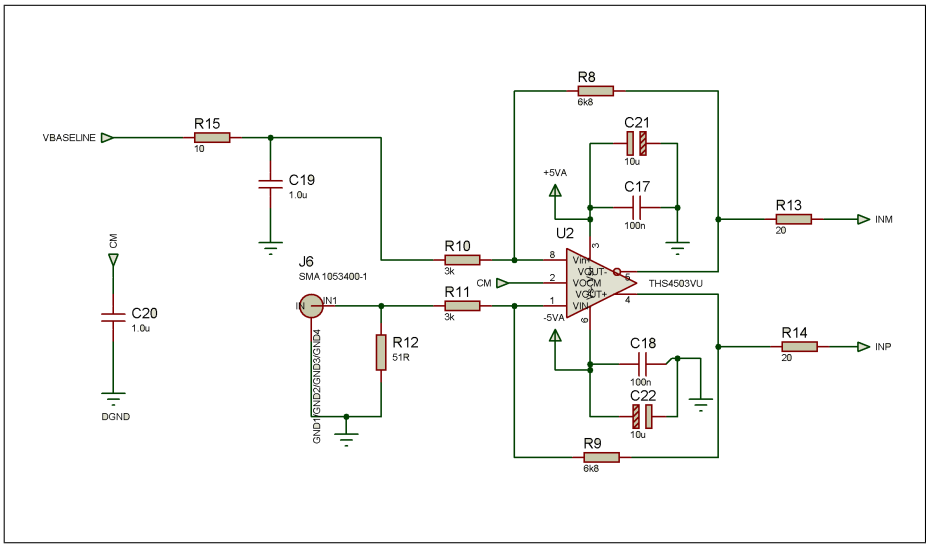
5.3. Bloques implementados

En esta sección se muestran los bloques de circuitos de la tarjeta de adquisición de datos LAGO-GT V0.2, con las correcciones de la versión V0.1, cada sub-sección detalla la electrónica implementada en cada bloque.

5.3.1. *Pulse Shapper*

La figura 40 muestra el circuito integrado THS4503 en configuración diferencial y la figura 41 se muestra la configuración del ADS5500.

Figura 40. **Amplificador diferencial utilizando el THS4503**



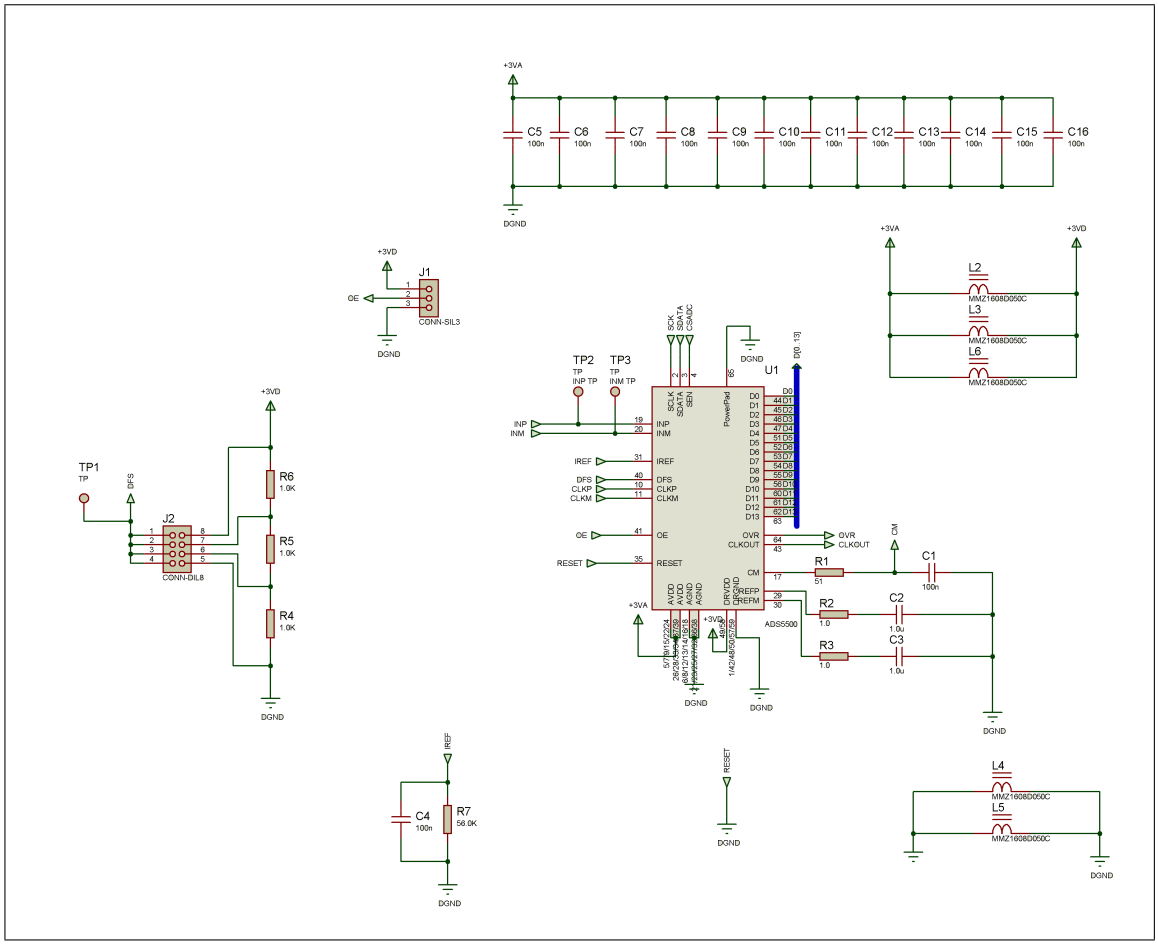
Fuente: elaboración propia, con el programa Proteus.

5.3.2. *Slow-control* y *baseline control*

La figura 42 muestra el traductor de voltaje LSF0108 utilizado para convertir la señal digital de $+3,3V.$ proveniente de la NEXYS 3 a una señal digital de $+5,0V.$

El traductor de voltaje se utiliza para manejar las señales de comunicación SPI en el resto de la electrónica. Las figuras 43 y 44 muestran la configuración del TLV5614 y del DAC7614 a utilizarse en el *slow-control* y el *baseline control*

Figura 41. **Configuración del ADS5500**

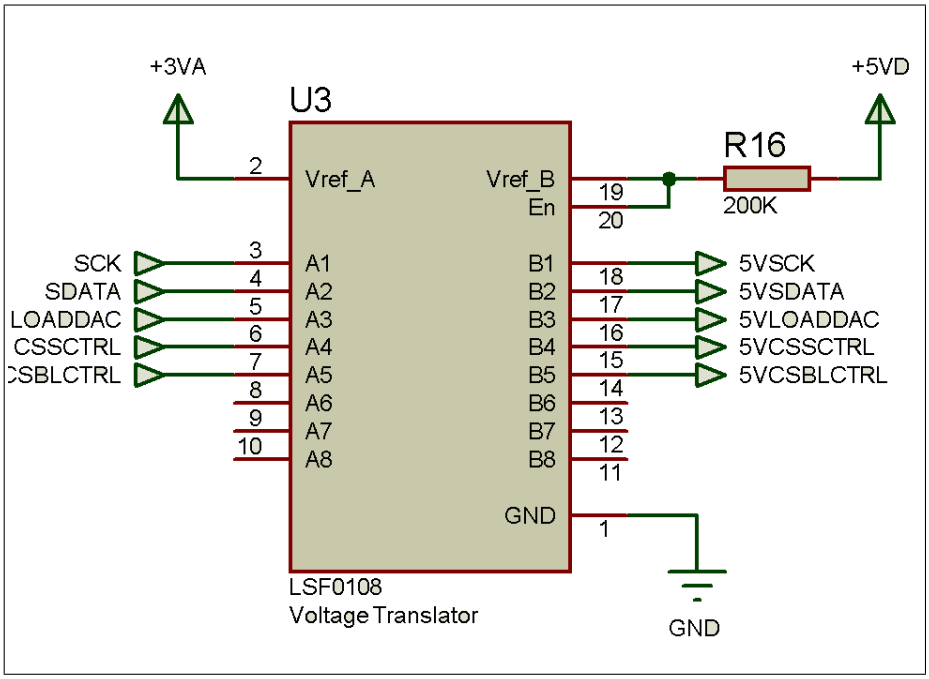


Fuente: elaboración propia, con el programa Proteus.

5.4. *PCB layout*

Un factor de suma importancia que fue pasado por alto en el prototipo inicial fue la complejidad del diseño del *layout*; ya que se está trabajando con electrónica de alta velocidad existen simplificaciones sobre la propagación de la onda y movimiento de electrones que no pueden despreciarse.

Figura 42. **Configuración del LSF0108, traductor de voltaje**

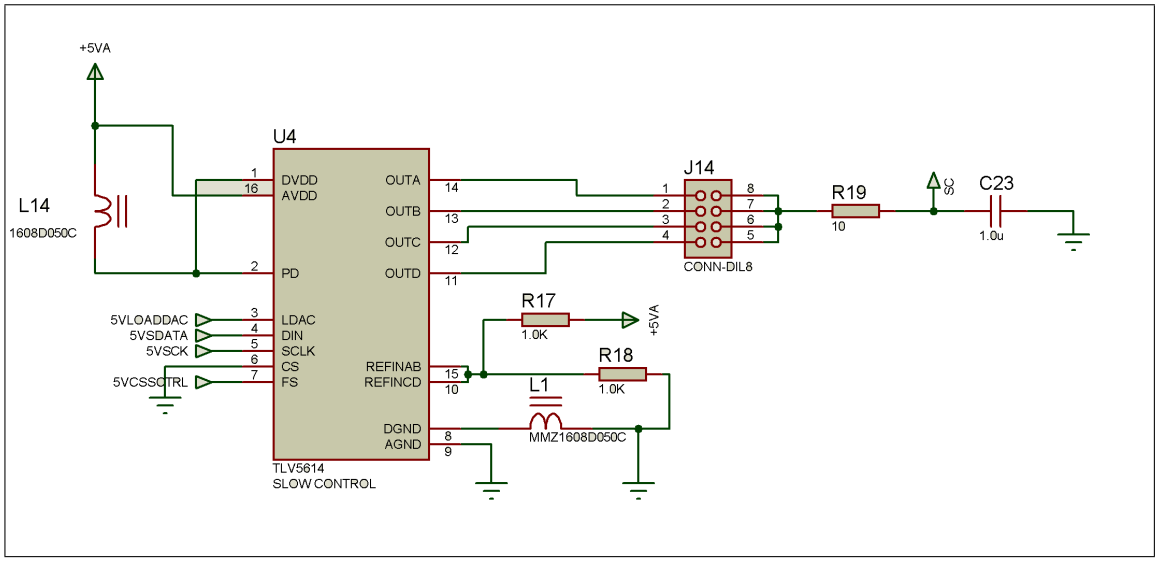


Fuente: elaboración propia, con el programa Proteus.

En la nota técnica *High-speed layout guidelines* se detallan las consideraciones a tomar en el diseño del *layout* de un PCB que trabaja con señales a alta velocidad. Las consideraciones a tomar son:

- Efectos del *crosstalk*
- Efectos del *skew*
- Alimentación
- Paneles de poder y de tierra
- Capacitores de desacople

Figura 43. **Configuración del TLV5614 para V_{Out} del *slow-control***



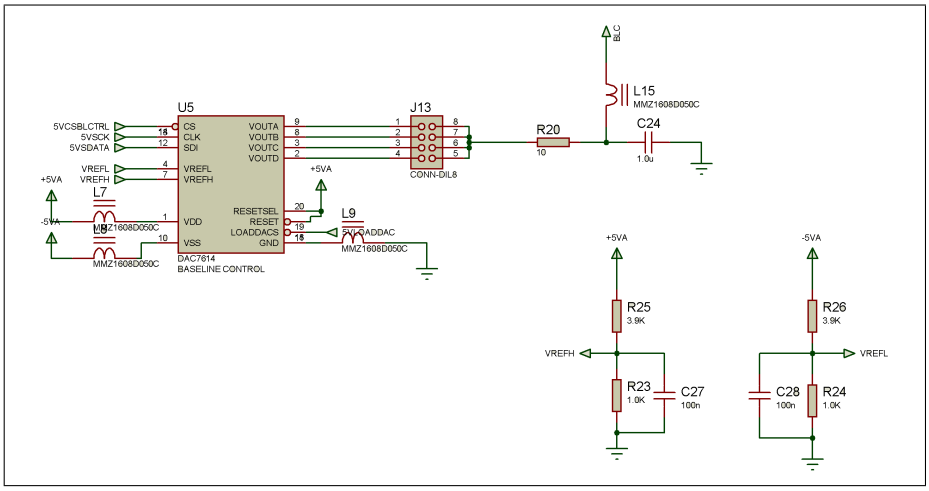
Fuente: elaboración propia, con el programa Proteus.

5.4.1. *Crosstalk* y *skew*

Es la influencia mutua que ocurre entre dos rutas paralelas. Una es llamada el agresor y la otra la víctima. Dado los campos electromagnéticos generados, el agresor induce una corriente inversa sobre la víctima provocando ruido. Para calcular el *crosstalk* se utilizó el programa ULTRACAD disponible de manera gratuita en la página http://www.ultracad.com/calc.htm.

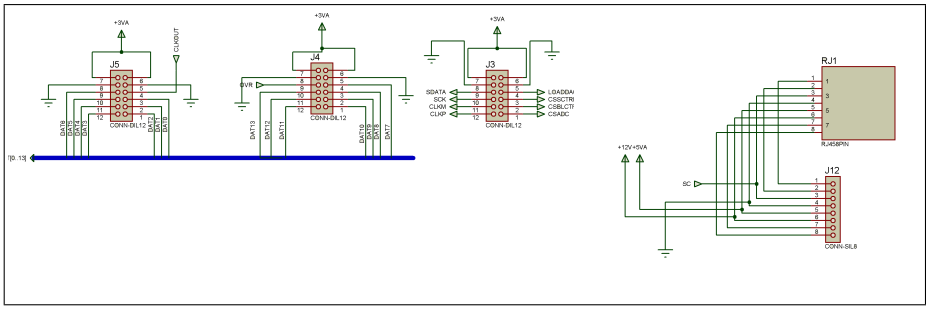
Para dos pistas paralelas con una longitud de 4,5 cm, un grosor de 10 mils, en una placa de fibra de vidrio a una separación de 10 mils entre cada pista, el coeficiente de *crosstalk* es de 0,148245. Para un voltaje digital de 3,3 voltios, la amplitud del crosstalk es de 0,49 voltios, lo cual ya es un error significativo.

Figura 44. **Configuración del DAC7614 para *baseline control***



Fuente: elaboración propia, con el programa Proteus.

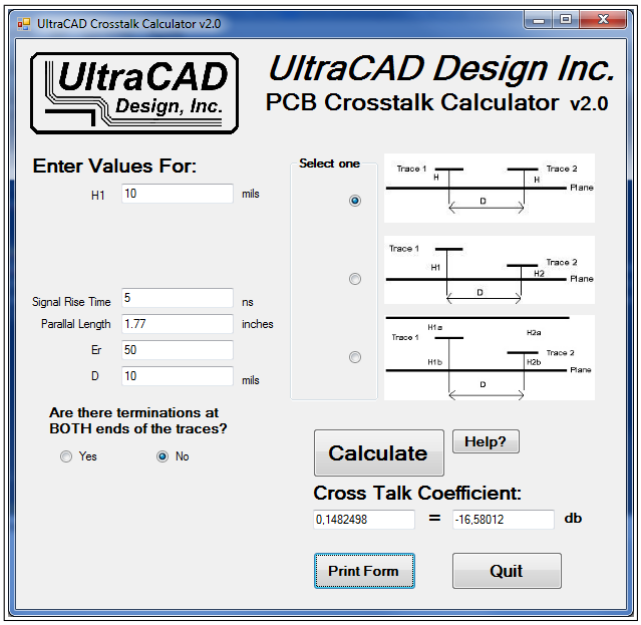
Figura 45. **Pines de salida y entrada de LAGOGTV0.2**



Fuente: elaboración propia, con el programa Proteus.

Skew es un retraso que ocurre en la salida paralalela de un circuito, este ocurre cuando la longitud de las pistas no es la misma a una salida. Si una pista es mas larga que la otra, las señales no llegan al mismo tiempo.

Figura 46. **Software ULTRACAD utilizado para medir el *crosstalk***



Fuente: elaboración propia.

El *skew* afecta, principalmente, cuando se utilizan señales diferenciales, por ejemplo, una señal de reloj, si una pista es mas larga que la otra, las señales pierden fase desestabilizando el sistema.

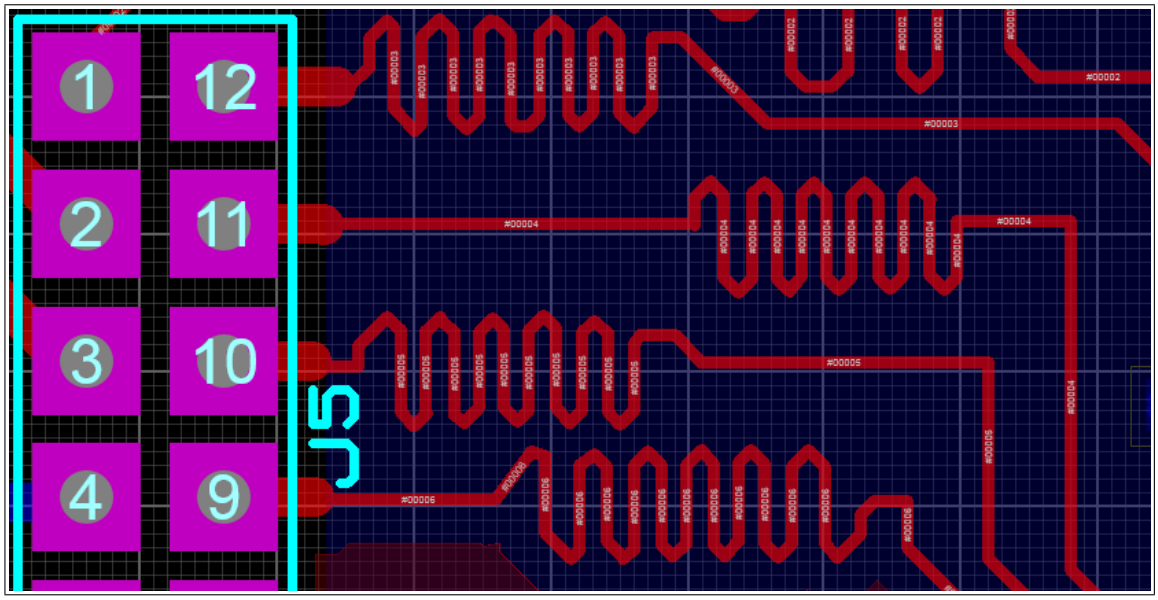
Ambos problemas pueden resolverse en el diseño del PCB, la nota técnica *Guidelines for Designing High-Speed FPGA PCBs* recomienda utilizar pistas diseñadas en serpentinas para igualar las longitudes en las salidas paralelas, en la figura 47 se muestra un ejemplo de pistas en serpentinas utilizadas en el diseño definitivo para la electrónica de LAGO-GT.

Cada pista en la salida paralela del ADS5500 hacia el conector tiene una longitud de 45 milimetros con una incerteza de $\pm$0,05mm. Las pistas en serpentina previenen además el efecto del *crosstalk*. Con el diseño de las pistas en serpentina la longitud en paralelo máxima es de 11,2 mm. lo cual genera un coeficiente de *crosstalk* máximo de 0,0054 lo que es equivalente 0,018 voltios p-p de amplitud del ruido. Este coeficiente fue calculado asumiendo el peor escenario posible, el cual es una señal digital alternante a 125 Mhz. La señal del reloj de sincronización del ADS5500 al encontrarse activo de manera permanente, es la principal fuente de *crosstalk* a prevenir, esto se logra enviando la señal por la cara posterior de la placa y separando el camino que toma la pista cuidando evitar traslape en el plano superior. El coeficiente calculado de *crosstalk* en estas consideraciones es de 0,017 implicando un voltaje parásito de 5 milivoltios. en la línea mas cercana (D0) el cual no es significativo al trabajar con señales digitales.

5.4.2. Alimentación y paneles de tierra

En electrónicas de alta velocidad los paneles de potencia se vuelven necesarios para asegurar la estabilidad del sistema. Lo ideal es que cada voltaje manejado tenga su propia capa y su propio panel de tierra, sin embargo, esto es in-práctico y costoso; ya que el proyecto es de bajo presupuesto, el máximo de capas que se disponen para el prototipo LAGO-GT V0.2 es de 2; por lo tanto, los paneles de poder se deben dividir. El problema de dividir paneles de tierra es la formación de ciclos de tierras o con un mal diseño puede comportarse como una antena e irradiar.

Figura 47. **PCB *layout* para corrección de *skew* y *crosstalk***

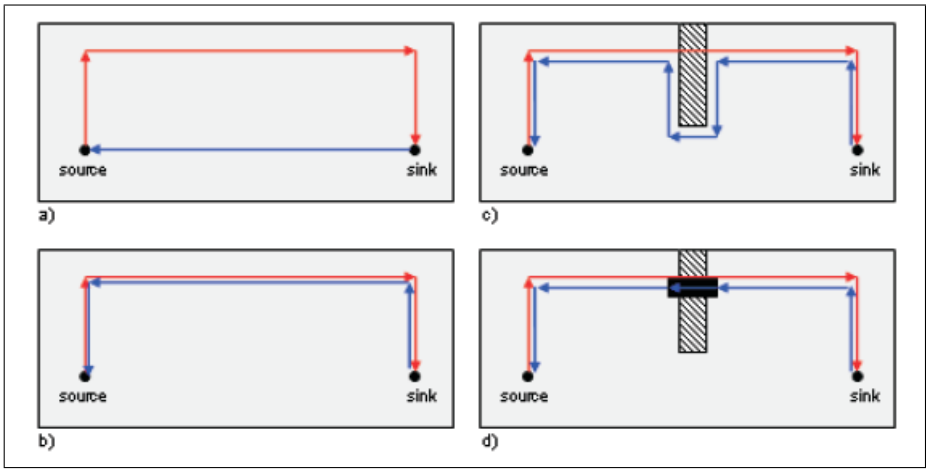


Fuente: elaboración propia, con el programa Proteus.

Para entender el concepto de los ciclos de tierra, hay que entender el comportamiento de los electrones. Los electrones siempre buscan el camino de la menor impedancia, ya que un circuito siempre es un camino cerrado, sin embargo, mientras mayor sea el área encerrada, mayor será la radiación emitida, esta radiación genera ruido e inestabilidad en el sistema.

La figura 48 muestra cuatro escenarios de las corrientes de retorno y las áreas producidas por cada una. En la figura 48a. se muestra el camino de retorno (línea azul) que toma el camino más corto; ya que la corriente proveniente de la fuente hasta donde se unifican los retornos encierra un área suficiente para radiar.

Figura 48. **Corriente de retorno y áreas resultantes**



Fuente: WEILER, A; PAKOSTA, A. *High-speed layout guidelines*. p. 18.

La figura 48 b muestra el camino de retorno que deben tomar los electrones, cuando hay diversos paneles de poder, el camino de retorno se alarga encerrando un área de radiación mayor, la figura 48 c muestra el camino de retorno ideal, siempre paralelo al circuito, con retornos inmediatos y cortos, minimizando el área de radiación al máximo, en el caso de paneles de poder, cuando se encuentran cortes y caminos hacia una tierra compartida como se observa en la figura 48 d, se deben colocar resistencias de 0 Ω para minimizar el ciclo, siempre buscando que el retorno sea lo más cercano posible al circuito.

Los paneles implementados en la electrónica de LAGO son los siguientes:

- +3,3 Voltios analógicos
- +5,0 Voltios analógicos
- -5,0 Voltios analógicos
- +5,0 Voltios digital

Figura 49. **Correcta e incorrecta disposición de vias en un capacitor**

Fuente: WEILER, A; PAKOSTA, A. *High-speed layout guidelines*. p. 23.

- Tierra digital DGND
- Tierra analógica

5.4.3. Capacitores de desacople

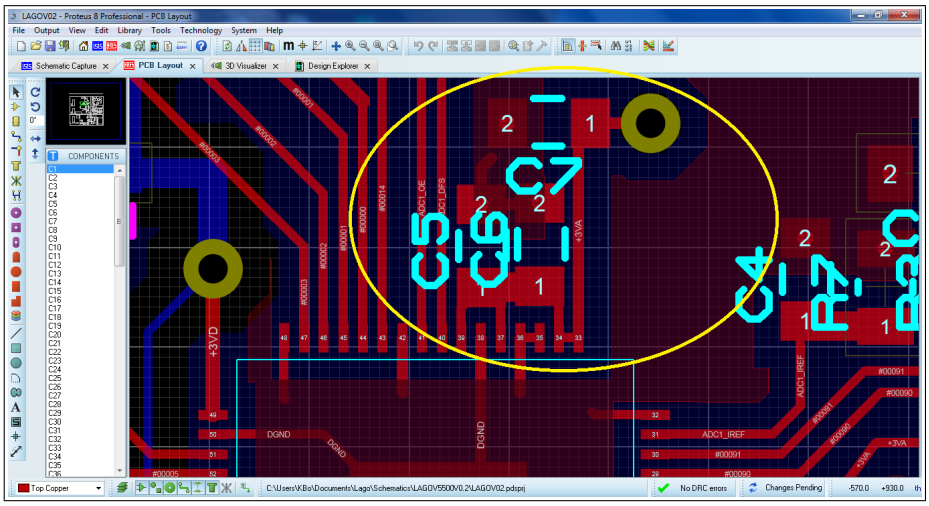
La distribución de los capacitores de desacople no puede ser al azar, existen ciertas condiciones que deben cumplirse para que el desacople sea efectivo.

- Colocar los capacitores de menor valor tan cerca como sea posible del dispositivo para minimizar la influencia inductiva del trazo.
- Colocar el capacitor de menor valor lo más cerca posible al pin o pines de poder.
- Conectar el extremo de capacitor a un panel de tierra de manera directa o a través de una vía.

La figura 49 muestra la configuración de una correcta y de una incorrecta disposición de vías para los capacitores de desacople.

Los capacitores de desacople utilizados en el ADS5500 se muestran en la figura 41 como capacitores C5 a C16 estos deben distribuirse los más cercano de cada uno de los pines de poder.

Figura 50. **Implementación de los capacitores de desacople en el ADS5500**

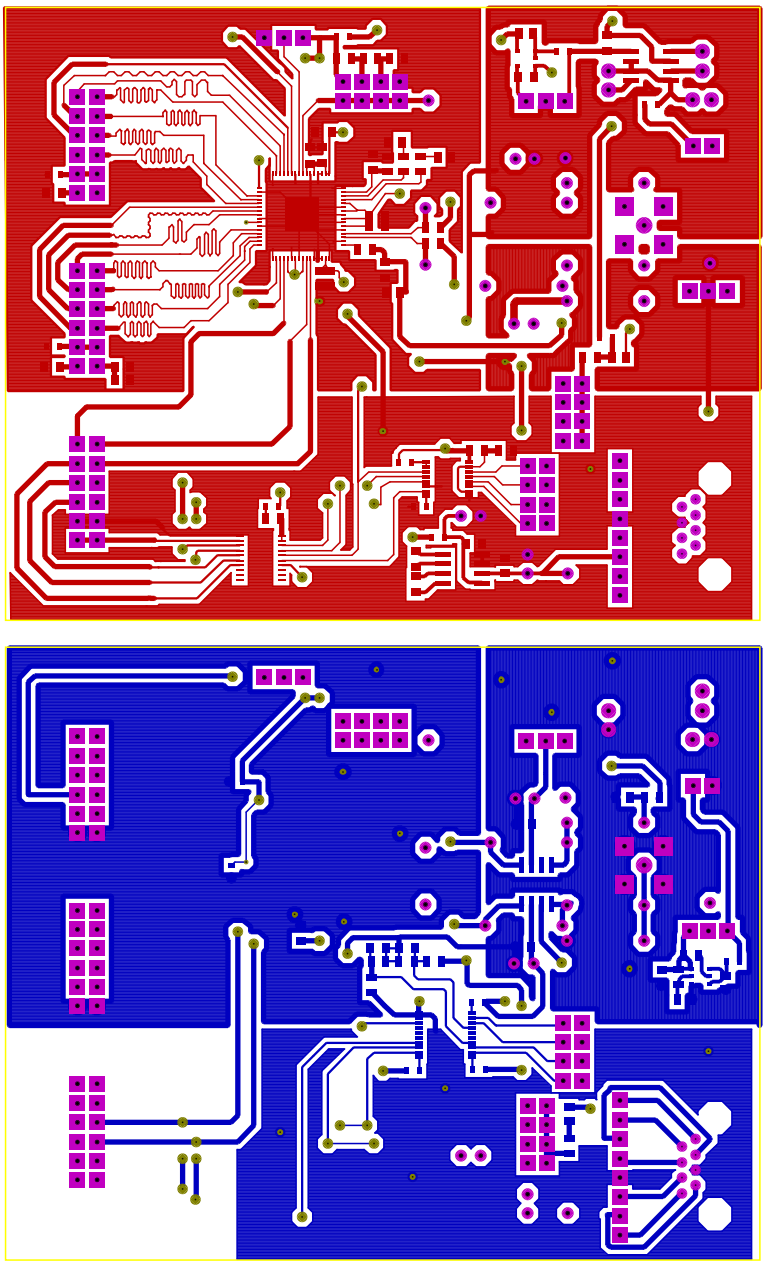


Fuente: elaboración propia, con el programa Proteus.

5.4.4. *Footprint* y fabricación del PCB

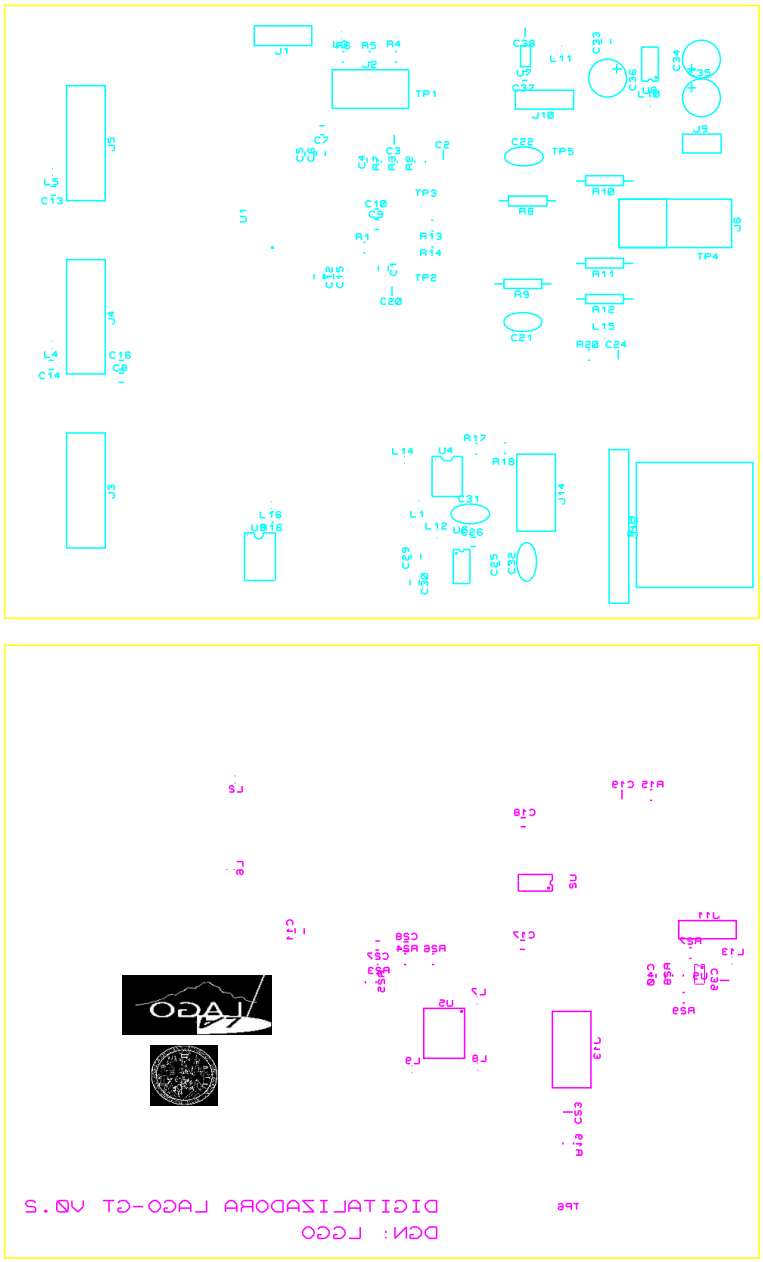
De la figura 51 a la 53 se detallan el *Footprint* de la PCB y su simulación 3D previo a su fabricación. En la figura 54 se puede observar el PCB del segundo prototipo de la tarjeta LAGO-GT ya fabricado.

Figura 51. **Capas de cobre superior e inferior para LAGO-GT V0.2**



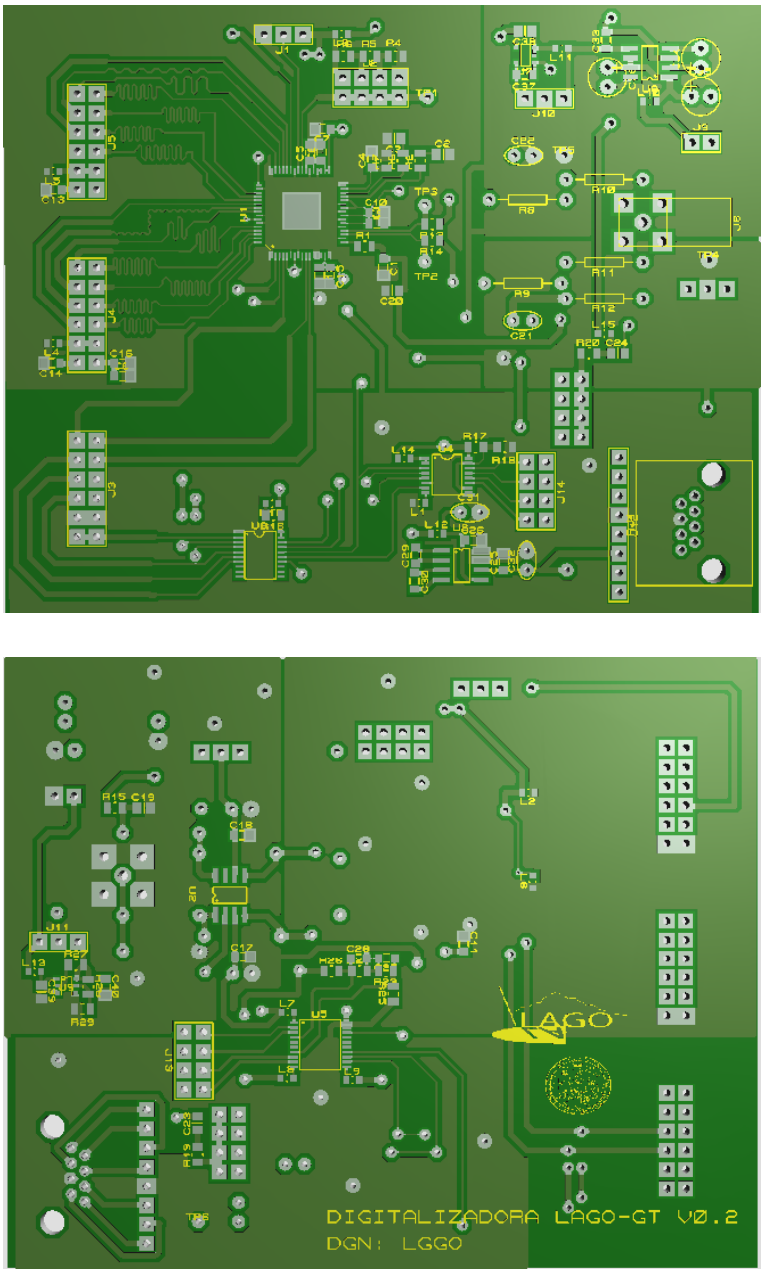
Fuente: elaboración propia, con el programa Proteus.

Figura 52. ***Silk screen* superior e inferior para LAGO-GT V0.2**



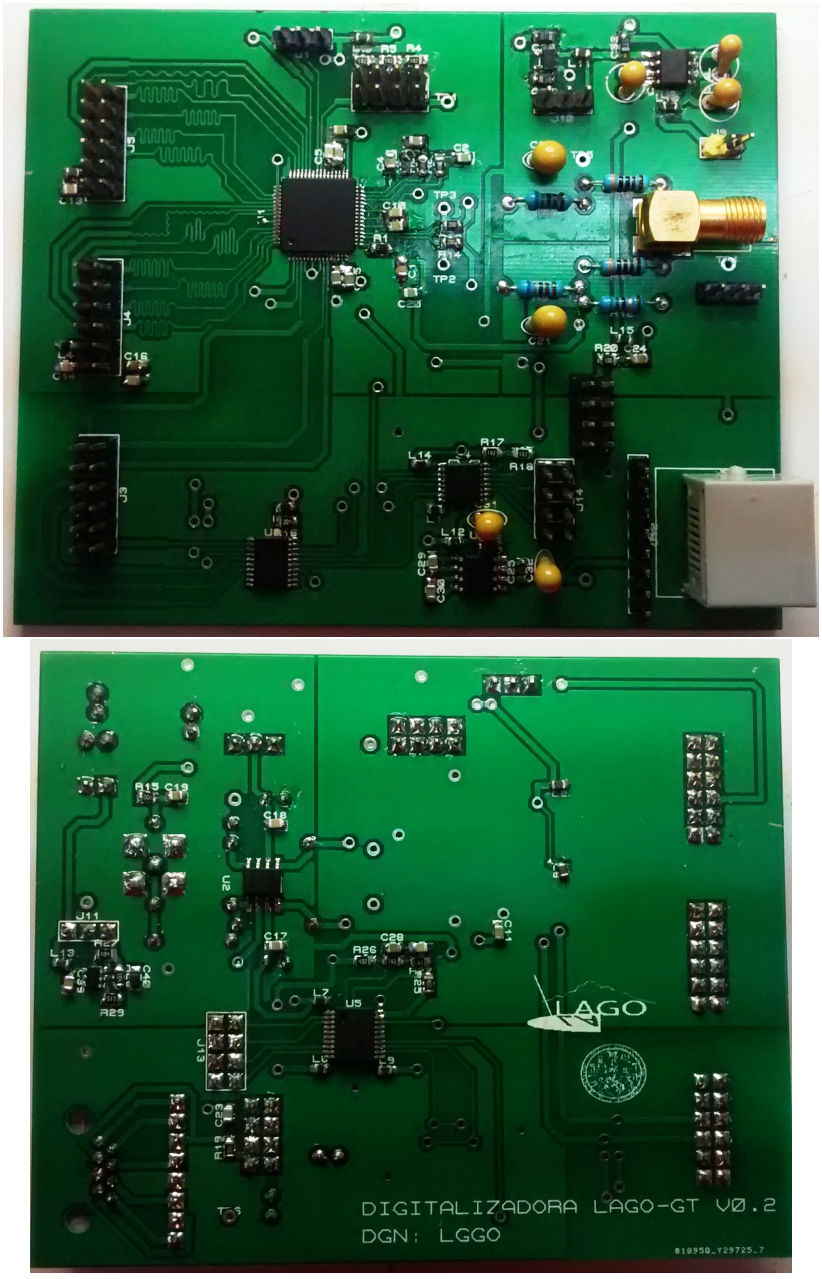
Fuente: elaboración propia, con el programa Proteus.

Figura 53. **Vista 3D de la PCB**



Fuente: elaboración propia, con el programa Proteus.

Figura 54. **PCB definitivo LAGO-GT V0.2**



Fuente: elaboración propia, con el programa Proteus.

6. BANCO DE PRUEBAS Y PLATAFORMA PARA *DEBUGGING*

Como se mencionó en capítulos anteriores, un prototipo debe de ser sometido a una serie de pruebas metódicas con el propósito de encontrar y corregir errores; este proceso es conocido como *debugging*.

Banco de pruebas se le llama al espectro de pruebas realizado en un prototipo o software simulando todas las condiciones de entrada para analizar su respuesta.

La diferencia principal entre el banco de pruebas y la plataforma de *debugging* consiste en que el banco de pruebas analiza únicamente la respuesta del sistema a diversas entradas observando el sistema como una caja negra. La plataforma de *debugging* usa un concepto de caja blanca para realizar las pruebas y corregir errores dentro del sistema.

En este caso, para probar el prototipo se necesita desarrollar ambas plataformas, el diseño de ambas es un tema bastante complejo en sí, por lo que en este capítulo se limita a la descripción del banco de pruebas y de la plataforma para *debugging*.

6.1. Plataforma para *debuging*

En esta plataforma se necesita controlar los valores de voltaje de salida del *baseline control*, del *slow-control*, del ADC que se controlan vía SPI. Se debe leer la salida diferencial del *pulse shapper* como referencia. Se necesita leer los valores de salida del ADC sincronizados con su reloj, controlar un valor de pulso vía PWM para excitar el PMT, retroalimentación visual de las pruebas que se están realizando y ocupar el menor número de periféricos posibles.

El control de los voltajes, la señal generada del PWM y la adquisición de los datos del ADC se realiza con la NEXYS 3. Los voltajes de salida del *baseline control* y del *slow-control*, así como la salida diferencial del *pulse shapper* se realiza midiendo en los puntos de prueba, con osciloscopio, las señales de salida.

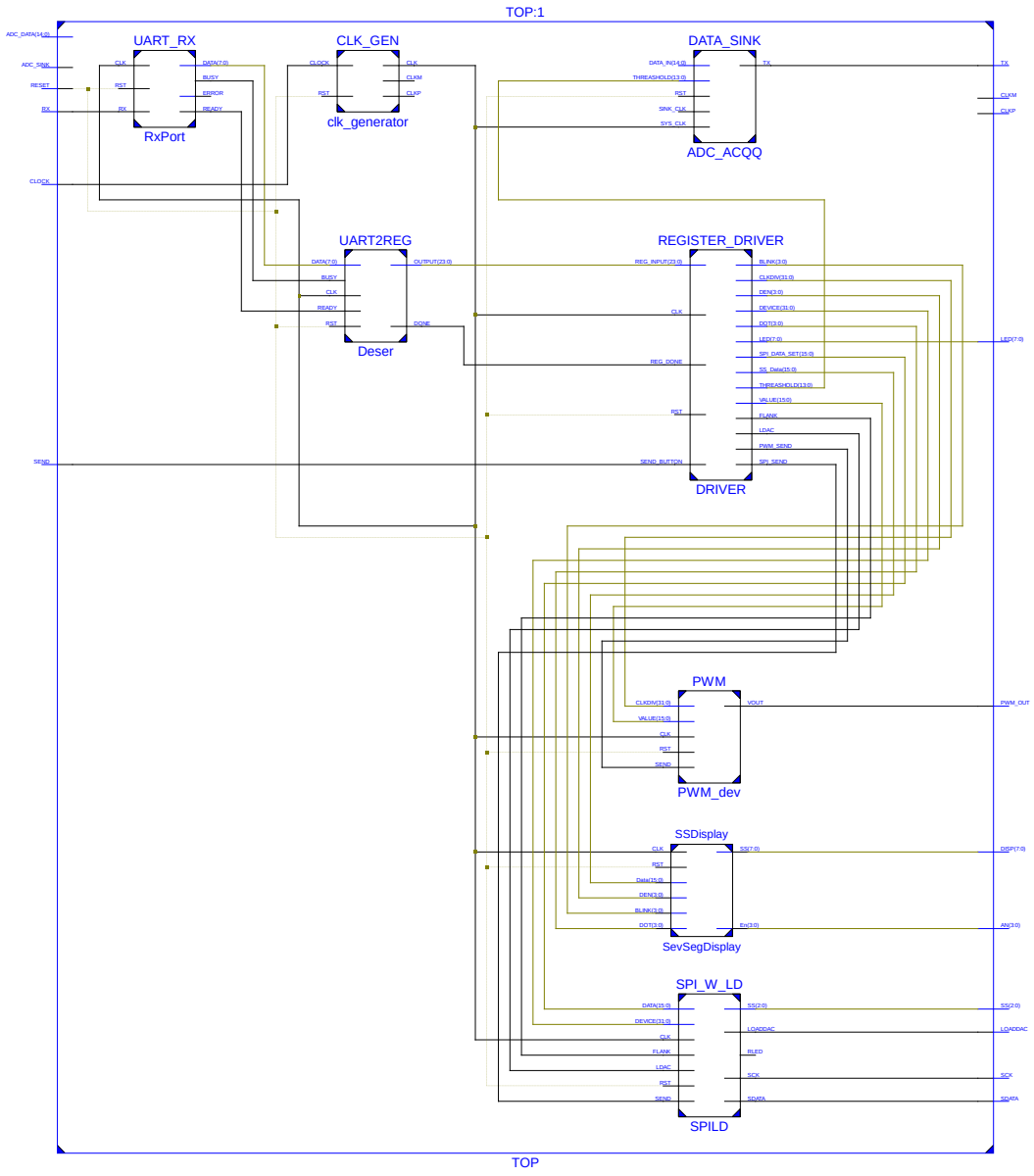
6.1.1. Componentes del VHDL de prueba

De manera similar a como se detalló el firmware de prueba en el capítulo 4. El firmware de *debugging* contiene los siguientes bloques:

- *Clock generator* encargado de generar el reloj diferencial para el ADC así como aislar el reloj del sistema.
- Comunicación serial para enviar las instrucciones de control, así como para recibir los datos provenientes del ADC.
- Bloque SPI para enviar las señales a los distintos bloques de hardware diseñados.
- Memoria FIFO para almacenaje de los datos de alta velocidad.
- Bloque PWM para generar pulsos de excitación al PMT.
- Display de siete segmentos, y LED's para la retroalimentación visual.

- Bloque de registros.

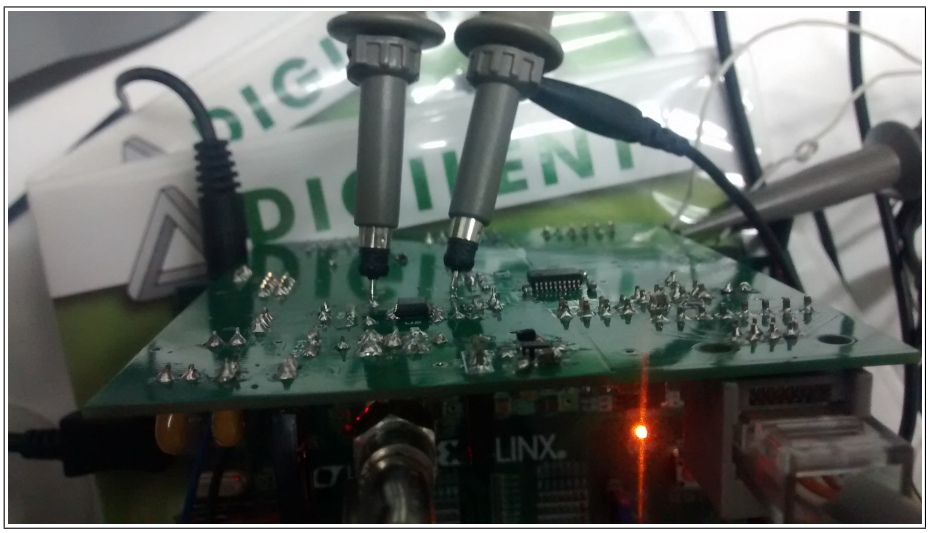
Figura 55. **Diagrama de bloques del VHDL**



Fuente: elaboración propia, con el programa *ISE Design Tools*.

En la figura 55 se muestra el diagrama de bloques implementado dentro de la FPGA el detalle de cada bloque y el código en VHDL se incluye en el apéndice.

Figura 56. **Conexión para *debugging* utilizada en las pruebas del PCB**



Fuente: elaboración propia.

Los puntos de prueba 2 y 3 fueron utilizados para medir la salida del amplificador diferencial. Utilizando la función interna *MATH*, dentro del osciloscopio, se puede observar la señal de entrada al ADC. Dado que la adquisición debe ser sincronizada con cada evento simulado por el PWM, el *trigger* se configura de forma externa con un nivel de 400 mV. conectado a la señal generadora del PWM (ver esquemáticos). La figura 56 muestra la configuración de las puntas de osciloscopio para poder replicar el evento.

6.1.2. Simulación de eventos

Para realizar las pruebas se diseñó en Python v. 2.7, una libreria de control para interactuar via serial con la FPGA. La librería provee comandos para montar valores del voltaje baseline, para el voltaje del *slow-control*, para controlar el ciclo de trabajo del PWM y para recibir los datos adquiridos después de un evento.

El protocolo para la comunicación de datos consiste en tres bytes. El primero de dirección de registro, y los otros dos de propósito general. En la ecuación 6.1. se expresa la trama de datos recibidos por la FPGA en donde A representa los bits de dirección; B representan bits de propósito general y X son bits de función.

$$\widehat{AAA}AXXXX + BBBBBBBB + BBBBBBBB \tag{6.1.}$$

Los bits de dirección representan:

- 0x1 para dirección SPI
- 0x2 para dirección de PWM
- 0x3 para configuración de *Threashold*
- El resto de bits se dejan abierto para incorporación de nuevas funciones

En el caso de las funcion SPI, las direcciónes de los bits de función son los siguientes:

- Modo continuo
- Dirección SPI
- Configuración del valor SPI

Tabla X. **Funciones disponibles en la libreria uif.py**

Métodos	**Descripción**
startUART(p, b)	Abre un puerto serial en el puerto *p* a una tasa de transferencia *b* baudios.
sendADC(val)	Envía el valor *val* al ADC.
sendSLC(val)	Envía el valor *val* al slow-control.
sendBLC(val)	Envía el valor *val* al baseline control.
changeVal(val)	Envía el valor *val* a la salida SPI activa
dinamicSPI(dev, on=True)	Activa el dispositivo *dev* (ADC, SLCTRL, BLCTRL) en modo dinámico; *on* determina si está encendido o apagado el modo dinámico.
PWM_STATE(on=True)	Enciende o apaga la señal del PWM según el estado de *on*
PWM_SET(val)	configura el valor del ciclo de trabajo del PWM
THR_SET(val)	Configura el valor del *Threashold*
getData(num)	Recupera *num* valores del puerto serial y los devuelve en un vector de num/2 valores.
getData1(num)	Recupera *num* valores del puerto serial y los devuelve en un vector de num/2 valores.
getData2(num)	Recupera *num* valores del puerto serial y los devuelve en un vector de num/2 valores.
getData3(num)	Recupera *num* valores del puerto serial y los devuelve en un vector de num/2 valores.
saveData(data)	Almacena el vector data en un archivo .csv en el escritorio.

Fuente: elaboración propia.

Para la dirección PWM, los bits de función disponibles son los siguientes:

- Modo continuo
- Configuración del ciclo de trabajo

En la libreria uif.py, descrita en apéndices, se plantea la clase *serIface* cuyos métodos se describen en la tabla X. Utilizando esta librería se simplifica la tarea de controlar los componentes SPI al enviar las órdenes vía serial.

Se realiza un barrido entre los valores hexadecimales 0x0000 al 0x0fff en el caso del voltaje baseline y entre los valores hexadecimales 0x8000 al 0x8fff para controlar el voltaje del *slow-control*. De esta forma se comprueba el funcionamiento de ambos DAC's.

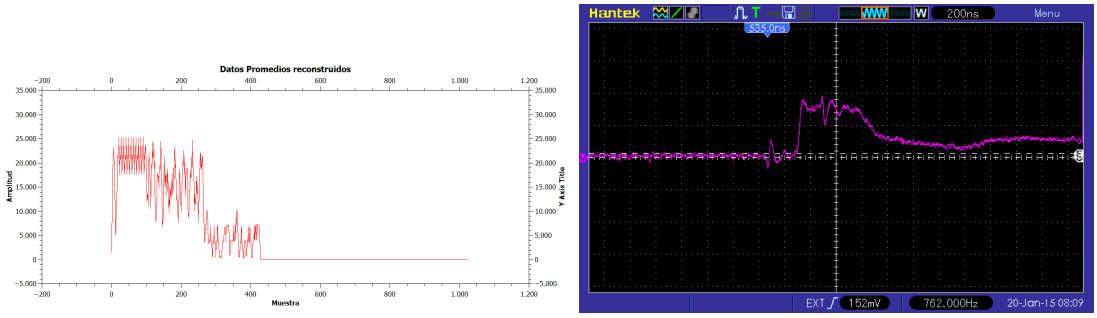
Se determina que el valor hexadecimal 0x07d0 (2000 en decimal) es el valor donde se encuentra en 0 el voltaje base. El voltaje de polarización óptimo es configurando el DAC del slow-control con el valor 0x83c0 (33728), el cual corresponde a un voltaje en el DAC de 1,04 voltios equivalente a 400 voltios en el PMT.

La FPGA entrega los datos provenientes del ADC con el siguiente formato:

$$0b10XXXXXXXXXXXXXX \tag{6.2.}$$

Donde X representa los 14 bits de datos provenientes del ADC. El 10 inicial se utiliza para indicar inicio de trama de datos y para chequeo de errores.

Figura 57. **Señal de entrada en el osciloscopio *versus* señal recibida**



Fuente: elaboración propia.

6.2. Banco de pruebas y resultados obtenidos

Para asegurarse que el funcionamiento del circuito es el adecuado, se pone a prueba cada componente serial la orden de fijar el voltaje de *baseline* a 0 V. Polarizar el PMT a 400 V. (el voltaje donde se observa el máximo valor del evento simulado) a través del *slow-control*. Una vez polarizado, se vacía el *buffer* del UART. El *Threashold* se coloca un 10 % arriba del valor del 0. Se lee la cola del buffer proveniente del UART; si hay cola, existe un error y se detiene el banco de pruebas. Ya aseguradas las condiciones de recepción, el banco de pruebas envía una señal para encender el PWM, el cual genera un impulso. Al pasar el umbral el programa espera recibir vía UART una trama de datos correspondiente al impulso, el cual es guardado en un archivo .csv para su posterior análisis. El *script* completo se detalla en el apéndice.

La figura 57 muestra la comparación entre un evento generado por el PWM, y la señal recibida por la tarjeta. Se aprecia la relación directa entre el evento entrante al ADC y la forma de la señal.

Al observar el cumplimiento del banco de pruebas en cada uno de sus pasos se puede garantizar un correcto funcionamiento de la tarjeta, en relación a sus salidas, la gráfica 57 es una representación de la respuesta de la tarjeta únicamente y no de una adquisición limpia que corresponde a un desarrollo más profundo del VHDL y software de control. Sin embargo, el hecho de que tenga un comportamiento estable ante un evento y que la relación de la salida sea directamente proporcional con la entrada, así como un cambio lineal en al leer los valores del *baseline* devolviendo un valor constante, garantiza el correcto funcionamiento del prototipo.

CONCLUSIONES

1. LAGO es un proyecto que pretende estudiar los destellos de radiación gamma (GRB), generados por eventos altamente energéticos y por efectos solares a través de distintos detectores Cherenkov localizados en Latinoamerica.

2. La tarjeta de adquisición LAGO-GT v0.2 presenta características superiores a las electrónicas desarrolladas actualmente para la adquisición de la forma del evento en el PMT.

3. Durante el proceso de diseño, el prototipo diseñado detalla los errores de diseño del circuito como del PCB a corregir en el circuito definitivo.

4. El prototipo LAGO-GT v0.2 puede adquirir una señal entrante del PMT con una amplitud máxima de 1.0V. , amplificarla según los valores de la resistencia de retroalimentación, y corregir un voltaje *offset* a través del THS4503.

5. La velocidad de muestreo máxima del prototipo LAGO-GT v0.2 es de 125 MSPS a través de un reloj diferencial con una resolución de 14 bits.

6. El voltaje de polarización del PMT tiene un voltaje de rizado menor a 25 milivoltios, el cual se logra por medio de filtros LDO e inductancias de choque, lo cual garantiza estabilidad en la polarización.

7. Los paneles de voltaje dan estabilidad al ruido y minimizan los efectos inductivos de las corrientes de retorno. Los capacitores de desacople y las inductancias de choque, así como los filtros LDO aíslan la propagación del ruido a través de las líneas de tierra.

8. Las características del prototipo diseñado permite un control del *slow control*, *baseline control* y el ADC a través de un bus SPI a una velocidad máxima de 20 MHz.

9. El prototipo LAGO-GT v0.2 presenta un funcionamiento lineal y directamente proporcional ante un impulso generado por un PWM, entregado a un LED ultravioleta, capturando la luz emitida en un PMT, adecuando la forma de la señal con respecto a un voltaje base y la forma de onda necesaria para ser muestreado a una tasa de 125 MSPS y 14 bits para reconstruir la señal.

10. A altas velocidades de muestreo es imperativo considerar el *skew* y el *crosstalk* como fuentes de ruido significativas en transmisiones en paralelo. Para evitar estos efectos, debe hacerse el diseño del PCB con pistas en serpentina con la misma longitud para las señales en paralelo.

RECOMENDACIONES

1. Considerar generar la señal de reloj diferencial utilizando la FPGA para garantizar la sincronía con el reloj de retorno.

2. Migrar al circuito integrado ADC3444, el cual tiene cuatro canales y un esquema de transmisión serial LVDS. Para esto tomar en cuenta el uso del protocolo VHDCI disponible en la tarjeta digitalizadora NEXYS 3.

3. Migrar a una tarjeta para la adquisición de datos FPGA con microprocesador embebido, sugerida una tarjeta ZYBO de Xilinx o una DE1-SOC de Altera para prescindir de un controlador externo.

4. Utilizar como software de control el lenguaje de programación Python 2.7, ya que presenta un nivel de abstracción alto y es compatible con cualquier plataforma Linux embebida.

5. Apoyar a nivel de universidad, de facultad y de escuela, el desarrollo de proyectos instrumentación con el propósito de investigación, ya que la falta de presupuesto limita el potencial humano disponible a nivel de estudiantado. Proyectos como LAGO no cuentan con apoyo económico por parte de la Universidad, por lo que su desarrollo depende únicamente de donaciones por terceras instituciones y muestras gratis.

BIBLIOGRAFÍA

1. Altera. *Guidelines for Designing High-Speed FPGA PCBs.* AN-315. EE.UU: 2004. 71 p.

2. BROWN, Burb. *Quad Serial Input 12-Bit, Voltage Output.* SBAS092. EE.UU: 1998. 17 p.

3. CONDE C, Rubén. *Sistema de adquisición para cuatro canales analógicos.* México: Universidad Autónoma de Puebla, LAGO-TN, 2012. 31 p.

4. GRUPEN, Claus; SHWARTZ, Boris. *Particle Detectors.* 2a ed. Cambridge, 1995. 677 p.

5. LIU, Hui-Quin. *14-bit, 125-MSPS ADS5500 evaluation.* SLY074. EE.UU: 2005. [en línea]: http://www.ti.com/lit/an/slyt074/slyt074.pdf [Consulta: mayo de 2014].

6. Maxim IC. *Simple Methods Reduce Input Ripple for All Charge Pumps* [en línea]: http://www.maximintegrated.com /en /app-notes/index.mvp/id/2027 [Consulta: mayo de 2014].

7. PÉREZ A, Yunior. *Caracterización de Detectores Cherenkov en el proyecto LAGO (Large Aperture GRB Observatory)*. Director: Dr. Luis Núñez. Universidad de los Andes. Departamento de Física, 2009. 215 p.

8. *Python.* [en linea] http://www.python.org. [Consulta: 30 de septiembre de 2014.]

9. SOFO HARO, Miguel. *LAGO Official Electronics guide.* Argentina: Centro Atómico Bariloche, 2011. 35 p.

10. Texas Instruments, *ADS5500/5541/5542/5520/5521/5522/ 14- and 12-Bit Single Chanel ADC EVM.* SLLU092. EE.UU: 2005. 35 p.

11. _____. *LSF010x 1/2/8 Channel Bidirectional Multi-Voltage Level Translator for Open-Drain and Push-Pull Application.* SDLS966D. EE.UU: 2013. 39 p.

12. _____. *12-Bit, 3 us Quad DAC, Serial Input, Programmable Setting Time, Low Power, H/W or S/W Power Down*. SBAS092. EE.UU: 2010. 24 p.

13. _____. *14-Bit, 125MSPS Analog to digital converter*. SBAS303F. EE.UU: 2008. 35 p.

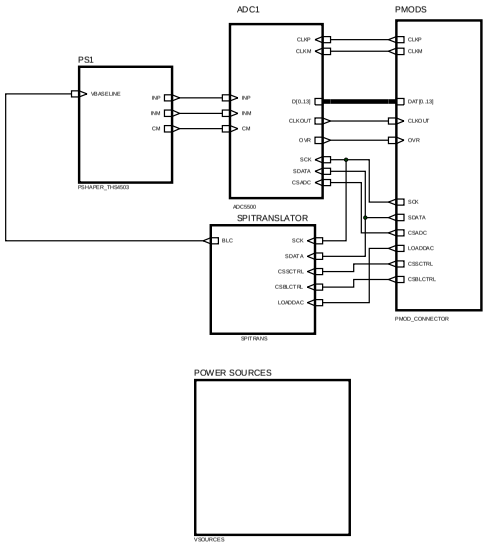
14. WEILER, Alexander; PAKOSTA, Alexander. *High-Speed Layout Guidelines.* SCAA082, EE.UU: 2006. 20 p.

APÉNDICES

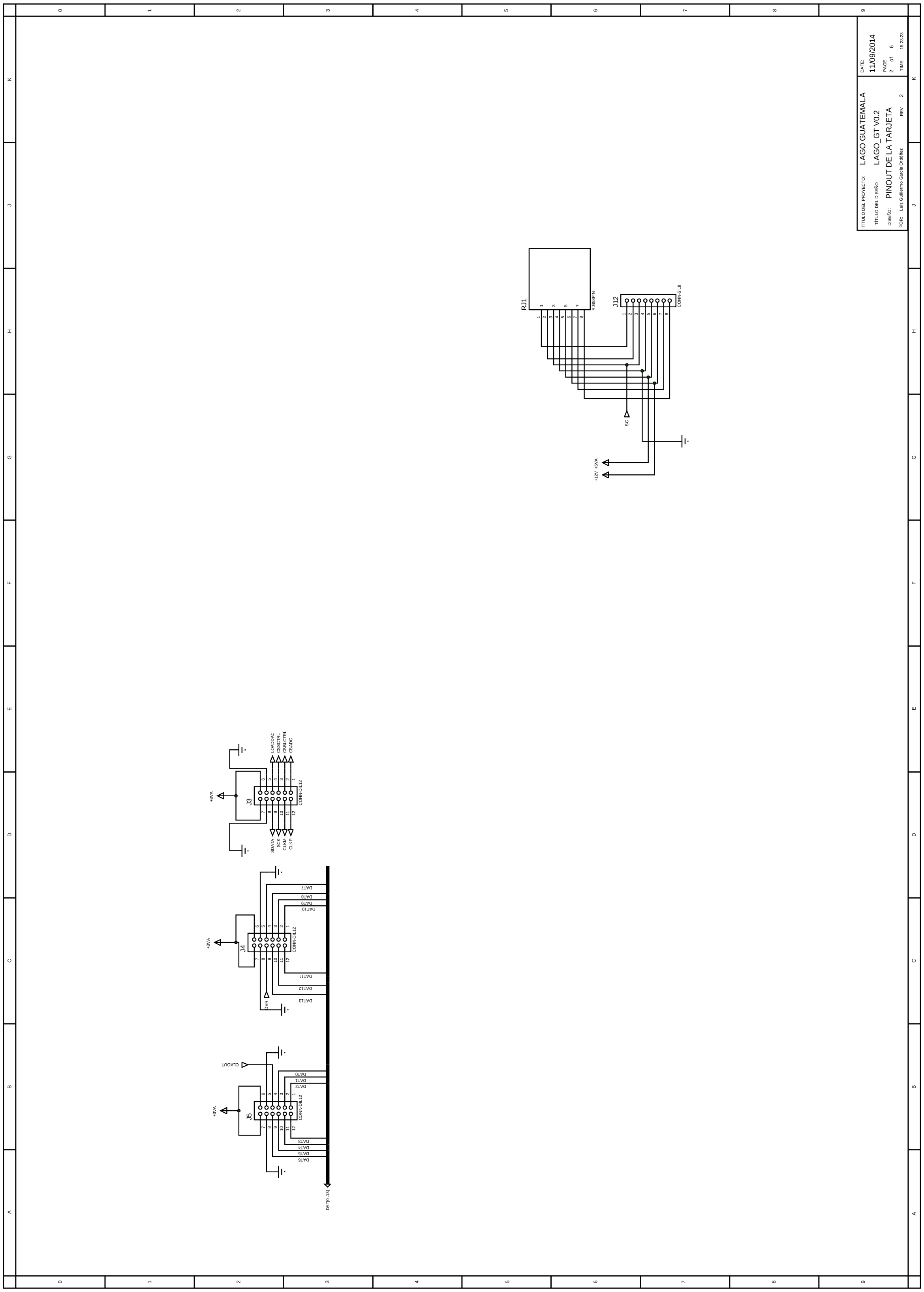
A. Esquemáticos de la tarjeta de adquisición de datos

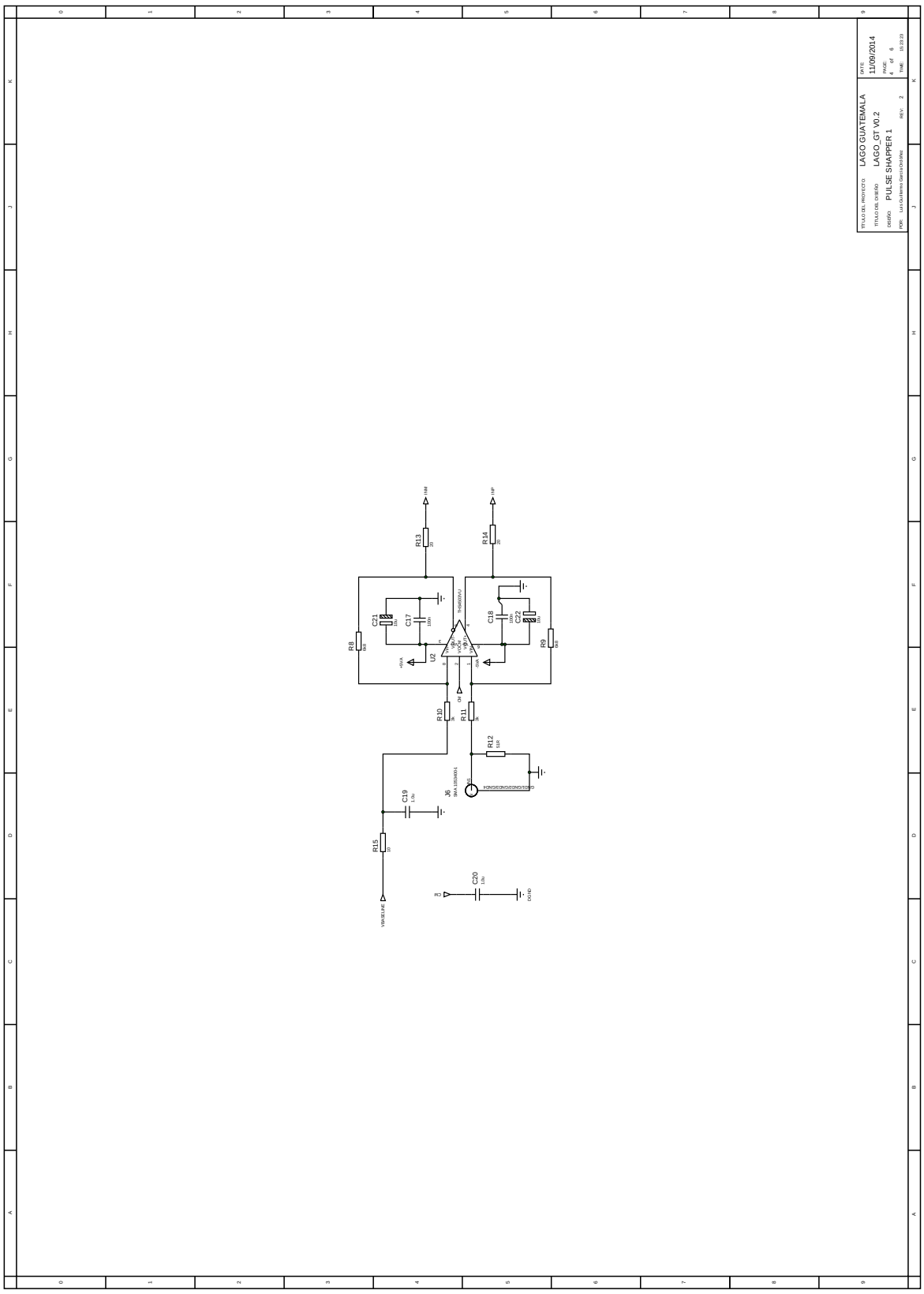
A continuación se presentan los esquemáticos del circuito implementado en el siguiente orden:

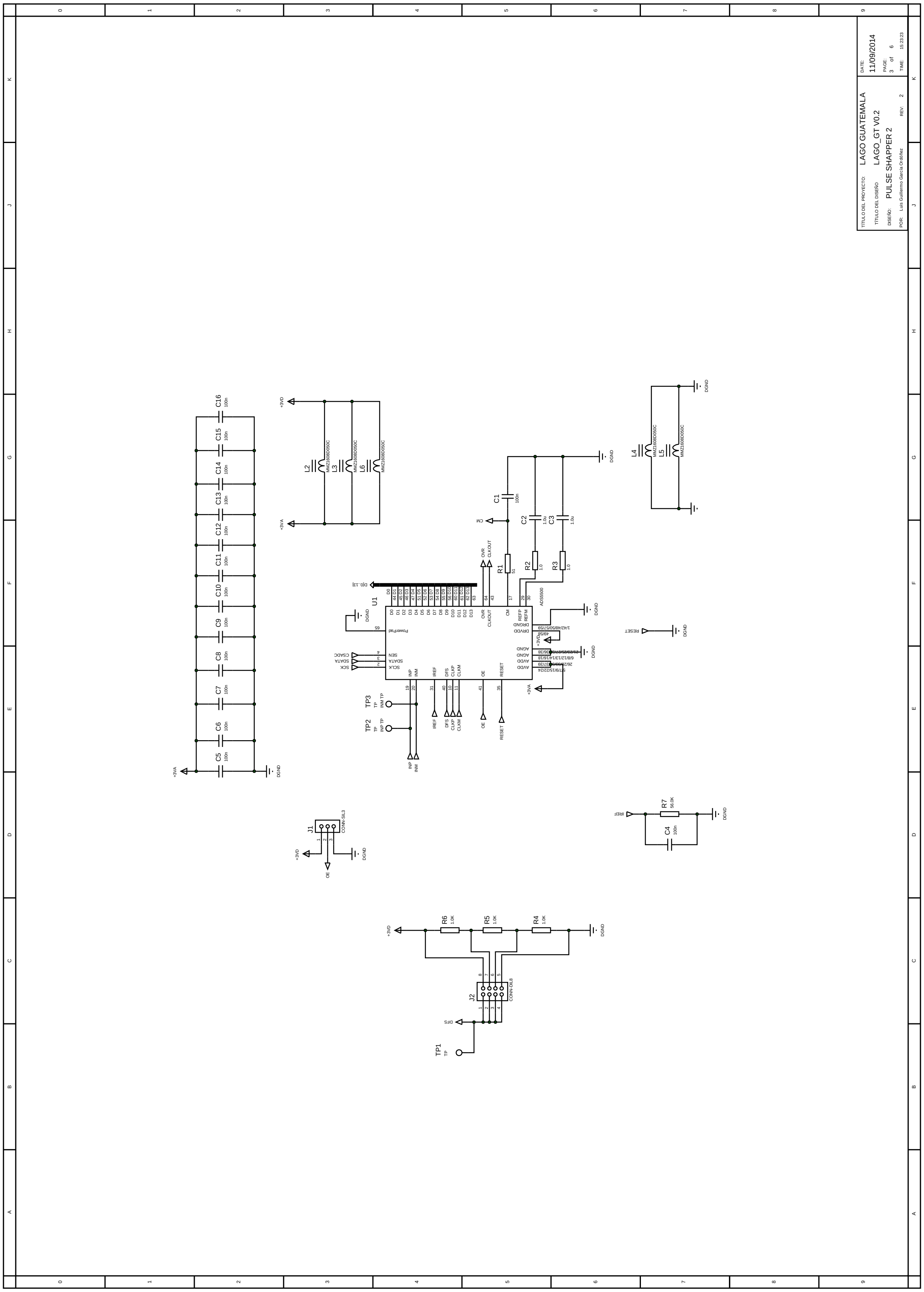
1. Diagrama de bloques
2. Pinout
3. *Pulse Shapper* 1
4. *Pulse Shapper* 2
5. Reguladora de voltaje
6. SCTRL y BLCTRL

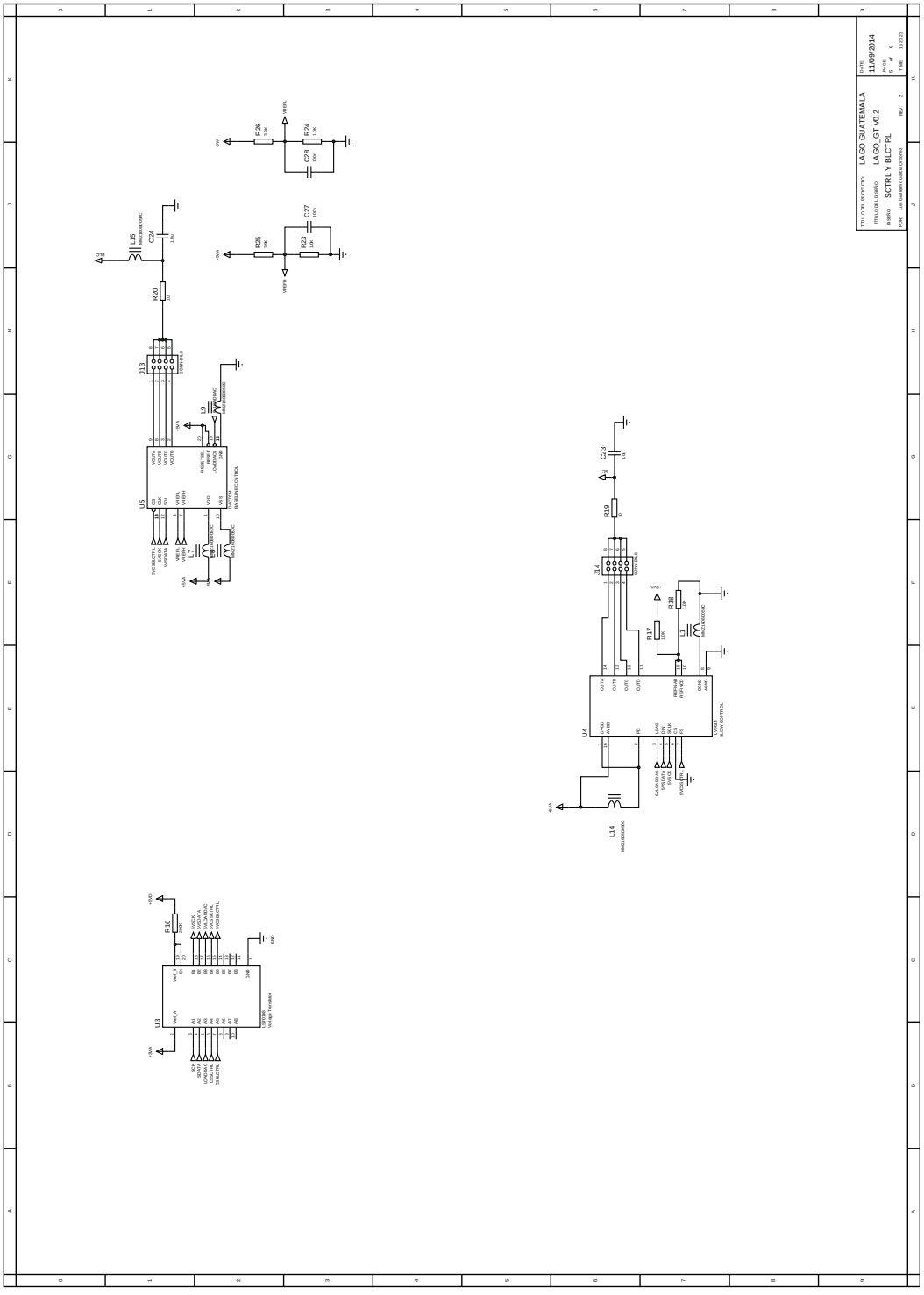


TITULO DEL PROYECTO:	LAGO GUATEMALA	DATE:	11/09/2014
TITULO DEL DISEÑO:	LAGO_GT V0.2	PAGE:	1 of 6
DISEÑO:	DIAGRAMA DE BLOQUES		
POR:	Luis Guillermo García Ordóñez	REV: 2	TIME: 15:23:23


LAGO GUATEMALA
LAGO_GT V0.2
PINOUT DE LA TARJETA
11/09/2014
RJ1
J12
J3
J4
J5


TITULO DEL PROYECTO LAGO GUATEMALA
TITULO DEL DISEÑO LAGO_GT V0.2
DISEÑO PULSE SHAPPER 1
DATE 11/09/2014
PAGE 4 of 6
REV 2
TIME 15:33:23
R13
R14
R8
R9
R10
R11
R12
R15
C17
C18
C19
C20
C21
C22
U2
J6
VHM
VHP
VBASELINE
DGND


LAGO GUATEMALA
LAGO_GT V0.2
PULSE SHAPPER 2
11/09/2014
U1
ADS5500



Fuente: elaboración propia utilizando el programa PROTEUS.

B. Bloques VHDL utilizados en el *firmware* de prueba LAGO-GT v 0.1

Modulo TOP

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    11:33:17 06/08/2014
-- Design Name:
-- Module Name:    Menu - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Menu is
        Generic(
                                slaves   : INTEGER :=3;                    -- SLAVE NUMBER
                                spiclkdiv        : INTEGER :=10; -- CLOCK DIVIDER
                                d_width  : INTEGER :=16;
                               -- Defining Menu Generic
                                Mclkdiv : INTEGER := 1000);
    Port ( CLK : in STD_LOGIC;
                           Left : in  STD_LOGIC;
           Right : in  STD_LOGIC;
           Up : in  STD_LOGIC;
           Down : in  STD_LOGIC;
           Enter : in  STD_LOGIC;
```

```
                SW : in STD_LOGIC_VECTOR(7 DOWNTO 0);
                ----Outputs----------
                ----Display----------
                segment7 : out STD_LOGIC_VECTOR (6 downto 0);
        LEN : out STD_LOGIC_VECTOR (3 downto 0);
                ----SPI----------
                SDATA : out STD_LOGIC;              -- SPI DATA OUTPUT
        SCK : out STD_LOGIC;                        -- SPI CLOCK
        SS : out STD_LOGIC_VECTOR (slaves-1 downto 0); --SLAVE SELECT
        LOADDAC : out STD_LOGIC;    -- LOAD DAC CONTROL
                LED    : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);    --LED DISPLAY
                ----DCM----------
                CLKP : OUT STD_LOGIC;
                CLKM : OUT STD_LOGIC
                );
end Menu;


architecture Behavioral of Menu is

component HexDisplay is
        Port ( Value : in STD_LOGIC_VECTOR (15 downto 0);
           Refresh : in STD_LOGIC;
           Seg : out STD_LOGIC_VECTOR (6 downto 0);
           DEN : out STD_LOGIC_VECTOR (3 downto 0);
           CLK : in STD_LOGIC;
           RST : in STD_LOGIC);
end component HexDisplay;

component SPIIFACE is
        Generic(
                        slaves  : INTEGER :=3;              -- SLAVE NUMBER
                        spiclkdiv       : INTEGER :=10; -- CLOCK DIVIDER
                        d_width  : INTEGER :=16);           -- DATA WIDTH
    Port ( CLK : in STD_LOGIC;                      -- SYSTEM CLOCK
           RST : in STD_LOGIC;                      -- RESET BUTTON
           DEVICE : in INTEGER;                     -- DEVICE NUMBER
           DATA : in STD_LOGIC_VECTOR (d_width-1 downto 0); --DATA VECTOR
           SEND : in STD_LOGIC;                     -- SEND SIGNAL
           SDATA : out STD_LOGIC;               -- SPI DATA OUTPUT
           SCK : out STD_LOGIC;                     -- SPI CLOCK
           SS : out STD_LOGIC_VECTOR (slaves-1 downto 0); --SLAVE SELECT
           LOADDAC : out STD_LOGIC;    -- LOAD DAC CONTROL
                   LED    : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));    --LED DISPLAY
end component SPIIFACE;

SIGNAL RST : STD_LOGIC;
SIGNAL PRESS : STD_LOGIC;
SIGNAL SELECTION : INTEGER RANGE 0 TO 2;

-- Defining state signals
TYPE state IS (RST_STATE, STDBY, ADC, BLCTRL, SCTRL, MSB, LSB, E_STATE);
signal curr_state, next_state: state;

-- Defining the Display signals.
```

```
signal disp_val : STD_LOGIC_VECTOR (15 DOWNTO 0);
signal disp_clk, disp_rst, disp_refresh : std_logic;
signal disp_seg7 : std_logic_vector (6 downto 0);
signal disp_len : std_logic_vector (3 downto 0);

-- Definition of SPI signals
signal spi_clk, spi_rst, spi_send: std_logic;
signal spi_device : INTEGER;
signal spi_data: std_logic_vector(15 downto 0);
signal spi_sdata, spi_sck, spi_loaddac: std_logic;
signal spi_ss : std_logic_vector (2 downto 0);
signal spi_led : std_logic_vector (7 downto 0);

-- Defining Button Signals

signal EnterS, UpS, DownS, LeftS, RightS : std_logic;
signal SEL : INTEGER RANGE 0 TO 2;

-- Defining a Buttons vector for ease of handeling Ther order is Enter, Up, Down, Left, Right
signal Buttons : std_logic_vector(4 downto 0);

begin

Disp:
component HexDisplay
        Port Map( Value                 => disp_val,
           CLK                  => disp_clk,
                        RST             =>      disp_rst,
                        Refresh         => disp_refresh,
           Seg  => disp_seg7,
           DEN          => disp_len
                        );

spi:
component SPIIFACE
         Generic map(
                                slaves          =>3,                    -- SLAVE NUMBER
                                spiclkdiv       =>10,   -- CLOCK DIVIDER
                                d_width         =>16
                                )
    Port map ( CLK      => spi_clk,
           RST  => spi_rst,
           DEVICE => spi_device,
           DATA         => spi_data,
           SEND         => spi_send,
           SDATA        => spi_sdata,
           SCK  => spi_sck,
           SS           => spi_ss,
           LOADDAC => spi_loaddac,
                        LED             => spi_led
                        );

-- RST SIGNAL IS WHEN UP, DOWN AND ENTER IS PRESSED AT THE SAME TIME.

RST<=UP AND DOWN AND ENTER;
```

```
-- GENERAL SIGNAL CONFIGURATIONS.
disp_rst<=RST;
spi_rst<=RST;
disp_clk<=clk;
spi_clk<=clk;
CLKP<=CLK;
CLKM<=NOT(CLK);

-- configuring outputs-----
segment7<=disp_seg7;
LEN<=disp_len;
SDATA<=spi_sdata;
SCK<=spi_sck;
SS<=spi_ss;
LOADDAC<=spi_loaddac;
--LED<=spi_led;


-- State changer
PROCESS(CLK, RST)
VARIABLE COUNT : INTEGER RANGE 0 TO Mclkdiv;
BEGIN
        IF RST='1' THEN
                curr_state <= RST_STATE;
                COUNT:=0;
        ELSIF RISING_EDGE(CLK) THEN
                If COUNT = Mclkdiv THEN
                        curr_state<=next_state;
                        COUNT:=0;
                ELSE
                        COUNT:=COUNT+1;
                END IF;
        END IF;
END PROCESS;

-- States definition
PROCESS(curr_state, LeftS, RightS, UpS, DownS, EnterS)
--variable sel : integer range 0 to 2:=0;
BEGIN
        --spi_device<=sel;
                CASE curr_state IS
                        WHEN RST_STATE=>
                                next_state<=STDBY;
                        WHEN STDBY=>
                                IF EnterS ='1' THEN
                                        next_state<=ADC;
                                ELSE
                                        next_state<=curr_state;
                                END IF;
                        WHEN ADC=>
                                IF UpS ='1' THEN
                                        next_state<=SCTRL;
                                ELSIF DownS ='1' THEN
                                        next_state<=BLCTRL;
```

```
                ELSIF EnterS='1' THEN
                        next_state<=LSB;
                        --sel:=2;
                ELSE
                        next_state<=curr_state;
                END IF;
            WHEN BLCTRL=>
                IF UpS ='1' THEN
                        next_state<=ADC;
                ELSIF DownS ='1' THEN
                        next_state<=SCTRL;
                ELSIF EnterS='1' THEN
                        next_state<=LSB;
                        --sel:=0;
                ELSE
                        next_state<=curr_state;
                END IF;
            WHEN SCTRL=>
                IF UpS ='1' THEN
                        next_state<=BLCTRL;
                ELSIF DownS ='1' THEN
                        next_state<=ADC;
                ELSIF EnterS='1' THEN
                        next_state<=LSB;
                        --sel:=1;
                ELSE
                        next_state<=curr_state;
                END IF;
            WHEN MSB=>
                IF UpS='1' THEN
                        next_state<= STDBY;
                ELSIF RightS='1' THEN
                        next_state<= LSB;
                ELSIF EnterS='1' THEN
                        next_state<= E_STATE;
                ELSE
                        next_state<=curr_state;
                END IF;
            WHEN LSB=>
                IF UpS='1' THEN
                        next_state<= STDBY;
                ELSIF LeftS='1' THEN
                        next_state<=MSB;
                ELSIF EnterS='1' THEN
                        next_state<= E_STATE;
                ELSE
                        next_state<=curr_state;
                END IF;
            WHEN E_STATE=>
                next_state<=STDBY;
            WHEN OTHERS=>
                next_state<=curr_state;
        END CASE;
END PROCESS;

-- KNOWING IF A BUTTON IS PRESSED
```

```
PRESS<=ENTER OR UP OR DOWN OR LEFT OR RIGHT;

EnterS<=Buttons(4);
UpS<=Buttons(3);
DownS<=Buttons(2);
LeftS<=Buttons(1);
RightS<=Buttons(0);

PROCESS(CLK, curr_state, RST, PRESS, UP, DOWN, LEFT, RIGHT, ENTER)
VARIABLE ispush : std_logic;
VARIABLE bstate : state;
VARIABLE COUNT : INTEGER RANGE 0 TO 10000;
BEGIN
        IF RST='1' THEN
                ispush:='0';
                Buttons<="00000";
                bstate:=curr_state;
                COUNT:=0;
        ELSIF RISING_EDGE(CLK) THEN
                IF PRESS='1' AND ispush='0' THEN
                        IF Enter='1' THEN
                                Buttons<="10000";
                        ELSIF Up='1' THEN
                                Buttons<="01000";
                        ELSIF Down='1' THEN
                                Buttons<="00100";
                        ELSIF Left='1' THEN
                                Buttons<="00010";
                        ELSIF Right='1' THEN
                                Buttons<="00001";
                        END IF;
                        ispush:='1';
                        bstate:=curr_state;
                ELSIF PRESS='1' AND ispush='1' AND  bstate /= curr_state THEN
                        Buttons<="00000";
                ELSIF PRESS='0' THEN
                        IF COUNT=10000 THEN
                                ispush:='0';
                                Buttons<="00000";
                                COUNT:=0;
                        ELSE
                                COUNT:=COUNT+1;
                        END IF;
                END IF;
        END IF;
END PROCESS;

-- DISPLAY CONTROL ACCORDING THE CURRENT STATE
PROCESS(curr_state, SW)
VARIABLE buff: STD_LOGIC_VECTOR(15 DOWNTO 0);
BEGIN
        CASE curr_state IS
                WHEN RST_STATE=>
                        buff:=(OTHERS =>'0');
                        disp_refresh <='0';
```

```
                disp_val<=x"0000";
                spi_data<=buff;
            WHEN STDBY=>
                disp_refresh <='1';
                buff:=(OTHERS =>'0');
                disp_val<=x"ABCD";
                spi_data<=buff;
            WHEN ADC=>
                buff:=x"0000";
                disp_val<=x"0ADC";
                disp_refresh <='1';
                spi_data<=buff;
            WHEN SCTRL=>
                buff:=x"0000";
                disp_val<=x"51C1";
                disp_refresh <='1';
                spi_data<=buff;
            WHEN BLCTRL=>
                buff:=x"0000";
                disp_val<=x"B1C1";
                disp_refresh <='1';
                spi_data<=buff;
            WHEN MSB =>
                buff:=buff AND x"00FF";
                buff(15 downto 8):=SW;
                disp_val<=buff;
                disp_refresh <='1';
                spi_data<=buff;
            WHEN LSB =>
                buff:=buff and x"FF00";
                buff(7 downto 0):=SW;
                disp_val<=buff;
                disp_refresh <='1';
                spi_data<=buff;
            WHEN E_STATE =>
                disp_val<=buff;
                disp_refresh <='1';
                spi_data<=buff;
            WHEN OTHERS=>
                buff:=x"0000";
                disp_val<=x"EEEE";
                spi_data<=buff;
        END CASE;
END PROCESS;

-- SPI DEVICE SELECTION AND SEND SIGNAL
spi_device<= 0 WHEN curr_state=RST_STATE else
                                0 WHEN curr_state=BLCTRL else
                                1 WHEN curr_state=SCTRL else
                                2 WHEN curr_state=ADC else
                                spi_device;

spi_send<= '1' WHEN curr_state=E_State else
                        '0';
```

```
-- REDUNDANCE LED
LED(2 DOWNTO 0)<= "001" WHEN spi_device=0 else
                                                "010" WHEN spi_device=1 else
                                                "100" when spi_device=2 else
                                                "111";

LED(7 downto 3)<= "00000";
end Behavioral;
```

Componente Hex Display

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    17:07:51 06/10/2014
-- Design Name:
-- Module Name:    HexDisplay - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity HexDisplay is
        GENERIC (CDIVIDER : INTEGER :=1000);
    Port ( Value : in  STD_LOGIC_VECTOR (15 downto 0);
           Refresh : in  STD_LOGIC;
           Seg : out  STD_LOGIC_VECTOR (6 downto 0);
           DEN : out  STD_LOGIC_VECTOR (3 downto 0);
           CLK : in  STD_LOGIC;
           RST : in  STD_LOGIC);
end HexDisplay;

architecture Behavioral of HexDisplay is
```

```
TYPE STATE IS (RSTSTATE, UM, UC, UD, UU, CLEAN);

signal curr_state, next_state : STATE;
signal DEN_SIG : STD_LOGIC_VECTOR (3 DOWNTO 0);
signal BCD, M, C, D, U : INTEGER RANGE 0 TO 16;
begin

PROCESS(CLK, RST)
VARIABLE COUNT : INTEGER RANGE 0 TO CDIVIDER :=0;
BEGIN
IF RST='1' THEN
        curr_state<=RSTSTATE;
elsif rising_edge(CLK) THEN
        IF COUNT = CDIVIDER THEN
                curr_state<=next_state;
                COUNT:=0;
        ELSE
                COUNT:=COUNT+1;
        END IF;
END IF;
END PROCESS;

PROCESS(curr_state, Value)
BEGIN
CASE curr_state IS
        WHEN RSTSTATE=>
                        next_state<=UM;
        WHEN UM=>
                next_state<=UC;
        WHEN UC=>
                next_state<=UD;
        WHEN UD=>
                next_state<=UU;
        WHEN UU=>
                next_state<=CLEAN;
        WHEN CLEAN=>
                next_state<=UM;
        WHEN OTHERS =>
                next_state<=RSTSTATE;
END CASE;
END PROCESS;

WITH curr_state SELECT
BCD<=   M WHEN UM,
                C WHEN UC,
                D WHEN UD,
                U WHEN UU,
          16 WHEN CLEAN,
                0 WHEN OTHERS;

PROCESS(RST,REFRESH, VALUE)
BEGIN
IF RST ='1' THEN
        M<=0;
        C<=0;
        D<=0;
```

```
        U<=0;
ELSIF Refresh='1' THEN
        M<=(to_integer(unsigned(Value)) MOD 65536)/4096;
        C<=(to_integer(unsigned(Value)) MOD 4096)/256;
        D<=(to_integer(unsigned(Value)) MOD 256)/16;
        U<=(to_integer(unsigned(Value)) MOD 16);
END IF;
END PROCESS;

WITH curr_state SELECT
DEN_SIG<= "1111" WHEN RSTSTATE,
                      "0111" WHEN UM,
                      "1011" WHEN UC,
                      "1101" WHEN UD,
                      "1110" WHEN UU,
                      "1111" WHEN OTHERS;

DEN<=DEN_SIG;

PROCESS(BCD)
BEGIN
case  bcd is
        when 0=> Seg <="1000000"; -- '0'
        when 1=> Seg <="1111001"; -- '1'
        when 2=> Seg <="0100100"; -- '2'
        when 3=> Seg <="0110000";       -- '3'
        when 4=> Seg <="0011001"; -- '4'
        when 5=> Seg <="0010010";       -- '5'
        when 6=> Seg <="0000010"; -- '6'
        when 7=> Seg <="1111000"; -- '7'
        when 8=> Seg <="0000000"; -- '8'
        when 9=> Seg <="0010000"; -- '9'
         --nothing is displayed when a number more than 9 is given as input.
        -- Distribution--------ABCDEFG---------
        when 10=> Seg <="0001000"; -- 'A'
        when 11=> Seg <="0000011"; -- 'B'
        when 12=> Seg <="1000110"; -- 'C'
        when 13=> Seg <="0100001"; -- 'D'
        when 14=> Seg <="0000110"; -- 'E'
        when 15=> Seg <="0001110"; -- 'F'
        when others=> Seg <="1111111";
end case;
END PROCESS;




end Behavioral;
```

Componente SPIIFACE

```
-- Company: UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
-- Engineer: LUIS GUILLERMO GARCIA ORDONEZ
--
-- Create Date:    22:58:23 01/24/2014
-- Design Name: SPI INTERFACE FOR LAGO-GUATEMALA ACQUISITION BOARD
-- Module Name:    TOP - Behavioral
-- Project Name: LAGO-GUATEMALA DIGITALIZER
-- Target Devices: ADS5500(DEVICE 2), TLV5614(16)(DEVICE 1), DAC7614 (DEVICE 0)
-- Tool versions:
-- Description:
-- THIS MODULE BRINGS A COMUNICATION INTERFACE FOR THE SPI DEVICES ON LAGO-GUATEMALA BOARD.

-- Works with SPIMaster module.
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity SPIIFACE is
        Generic(
                                slaves  : INTEGER :=3;                  -- SLAVE NUMBER
                                spiclkdiv       : INTEGER :=10; -- CLOCK DIVIDER
                                d_width  : INTEGER :=16);               -- DATA WIDTH
    Port ( CLK : in  STD_LOGIC;                                 -- SYSTEM CLOCK
           RST : in  STD_LOGIC;                                 -- RESET BUTTON
           DEVICE : in  INTEGER;                                -- DEVICE NUMBER
           DATA : in  STD_LOGIC_VECTOR (d_width-1 downto 0); --DATA VECTOR
           SEND : in  STD_LOGIC;                                -- SEND SIGNAL
           SDATA : out  STD_LOGIC;                      -- SPI DATA OUTPUT
           SCK : out  STD_LOGIC;                                -- SPI CLOCK
           SS : out  STD_LOGIC_VECTOR (slaves-1 downto 0); --SLAVE SELECT
           LOADDAC : out  STD_LOGIC;    -- LOAD DAC CONTROL
                        LED     : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));   --LED DISPLAY
end SPIIFACE;

architecture Behavioral of SPIIFACE is

COMPONENT spi_master IS
  GENERIC(
    slaves  : INTEGER := slaves;  --number of spi slaves
    d_width : INTEGER := d_width); --data bus width
```

```
  PORT(
    clock    : IN    STD_LOGIC;                              --system clock
    reset_n  : IN    STD_LOGIC;                              --asynchronous reset
    enable   : IN    STD_LOGIC;                              --initiate transaction
    cpol     : IN    STD_LOGIC;                              --spi clock polarity
    cpha     : IN    STD_LOGIC;                              --spi clock phase
    cont     : IN    STD_LOGIC;                              --continuous mode command
    clk_div  : IN    INTEGER;                                --system clock cycles per 1/2 period of sclk
    addr     : IN    INTEGER;                                --address of slave
    tx_data  : IN    STD_LOGIC_VECTOR(d_width-1 DOWNTO 0);   --data to transmit
    miso     : IN    STD_LOGIC;                              --master in, slave out
    sclk     : inout STD_LOGIC;                              --spi clock
    ss_n     : inout STD_LOGIC_VECTOR(slaves-1 DOWNTO 0);    --slave select
    mosi     : OUT   STD_LOGIC;                              --master out, slave in
    busy     : OUT   STD_LOGIC;                              --busy / data ready signal
    rx_data  : OUT   STD_LOGIC_VECTOR(d_width-1 DOWNTO 0)); --data received
END COMPONENT spi_master;

signal spiphase, spiflank, spicontmod, greset, spien : std_logic;
signal spiaddr : INTEGER;
signal spidata : STD_LOGIC_VECTOR(d_width-1 DOWNTO 0);

--Definiendo las salidas del spi
signal spiclock : std_logic;
signal slavesel : std_logic_vector (slaves-1 downto 0);
signal sdat : std_logic;
signal busysignal : std_logic;
signal test_signal: std_logic;


TYPE ESTADO IS (STDBY, WAIT1, WAIT2, WAIT3, ENVIA);

signal w1out, w2out, w3out : std_logic;

signal next_state, curr_state : ESTADO;
begin

--Informacion del SPI y sus condiciones segun dispositivo.
--BASESEL                DEVICE 0
--SLOWCTRL               DEVICE 1
--ADC                            DEVICE 2

spiaddr<=DEVICE;


--Definiendo el flanco de control del spi.
-- DEVICE 0 SPI_DATA_READ ON RISING_EDGE
-- ELSE SPI_DATA_READ ON FALLING_EDGE

-- Modificado 24/01/2014 22:12
--spiflank<= '0' when device = 0 else
--                      '1';
spiphase<= '1' when device = 0 else
                      '0';
-- Fin Modificacion
```

```
--Definiendo la fase en 0 y modo continuo de envio.
-- Modificado 24/01/2014 22:12
--spiphase<='1';
spiflank<='1';
-- Fin Modificacion
spicontmod<='0';


spi_1: spi_master
        PORT MAP(
          clock           => CLK,
    reset_n               => not(greset),
    enable                =>       spien,
    cpol          => spiflank,
    cpha          => spiphase,
    cont          => spicontmod,
    clk_div       => spiclkdiv,
    addr          => spiaddr,
    tx_data       => spidata,
    miso          => '0',
    sclk          => spiclock,
    ss_n          => slavesel,
    mosi          => SDAT,
    busy          => busysignal,
    rx_data               => open
         );

-- SPI OUTPUTS
spidata<=DATA;
SDATA<=SDAT;
SCK<=spiclock;
SS<=slavesel;
LED(0)<=busysignal;
LED(7 DOWNTO 1)<="0000000";


PROCESS(CLK, RST, next_state)
BEGIN
        IF RST='1' THEN
                greset<='1';
                curr_state<=stdby;
        elsif rising_edge(clk) then
                greset<='0';
                curr_state<=next_state;
        end if;
END PROCESS;

PROCESS(CURR_STATE, SEND, DEVICE, W1OUT, W2OUT, W3OUT, BUSYSIGNAL)
BEGIN
        CASE CURR_STATE IS
                WHEN STDBY =>
                        IF SEND='1' AND BUSYSIGNAL='0' THEN
                                IF DEVICE=0 OR DEVICE=1 THEN
                                        NEXT_STATE<=WAIT1;
```

```
                        ELSE
                                NEXT_STATE<=ENVIA;
                        END IF;
                ELSE
                        NEXT_STATE<=STDBY;
                END IF;
        WHEN WAIT1 =>
                IF W1OUT='1' then
                        NEXT_STATE<=ENVIA;
                ELSE
                        NEXT_STATE<=WAIT1;
                end if;
        WHEN ENVIA =>
                        IF DEVICE=0 OR DEVICE=1 THEN
                                NEXT_STATE<=WAIT2;
                        ELSE
                                NEXT_STATE<=STDBY;
                        END IF;
        WHEN WAIT2 =>
                IF W2OUT='1' then
                        NEXT_STATE<=WAIT3;
                ELSE
                        NEXT_STATE<=WAIT2;
                END IF;
        WHEN WAIT3 =>
                IF W3OUT='1' THEN
                        NEXT_STATE<=STDBY;
                ELSE
                        NEXT_STATE<=WAIT3;
                END IF;
        END CASE;
END PROCESS;

PROCESS(CURR_STATE, CLK, BUSYSIGNAL)
variable counter :integer;
BEGIN
if rising_edge(CLK) then
        CASE CURR_STATE IS
                WHEN STDBY =>
                        W1OUT<='0';
                        W2OUT<='0';
                        W3OUT<='0';
                        counter:=0;
                        spien <='0';
                        LOADDAC<='0';
                WHEN WAIT1 =>
                        LOADDAC<='1';
                        if counter=spiclkdiv then
                                counter:=0;
                                W1OUT<='1';
                        ELSE
                                counter:=counter+1;
                        END IF;
                WHEN ENVIA =>
                        spien <='1';
                WHEN WAIT2 =>
```

```
                        spien <='0';
                        IF COUNTER=spiclkdiv then
                                counter:=0;
                                W2OUT<='1';
                        ELSIF BUSYSIGNAL='0' THEN
                                COUNTER:=COUNTER+1;
                        END IF;
                WHEN WAIT3 =>
                        LOADDAC<='0';
                        IF COUNTER=spiclkdiv then
                                counter:=0;
                                W3OUT<='1';
                        ELSIF BUSYSIGNAL='0' THEN
                                COUNTER:=COUNTER+1;
                        END IF;
        END CASE;
end if;
END PROCESS;


end Behavioral;
```

Sub componente SPIMaster

```
----------------------------------------------------------------------------------
--
--  FileName:           spi_master.vhd
--  Dependencies:       none
--  Design Software:    Quartus II Version 9.0 Build 132 SJ Full Version
--
--  HDL CODE IS PROVIDED "AS IS."  DIGI-KEY EXPRESSLY DISCLAIMS ANY
--  WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT
--  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
--  PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL DIGI-KEY
--  BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL
--  DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF
--  PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
--  BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
--  ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.
--
--  Version History
--  Version 1.0 7/23/2010 Scott Larson
--    Initial Public Release
--  Version 1.1 4/11/2013 Scott Larson
--    Corrected ModelSim simulation error (explicitly reset clk_toggles signal)
--
----------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY spi_master IS
```

```
  GENERIC(
    slaves   : INTEGER := 3;  --number of spi slaves
    d_width  : INTEGER := 16); --data bus width
  PORT(
    clock    : IN     STD_LOGIC;                             --system clock
    reset_n  : IN     STD_LOGIC;                             --asynchronous reset
    enable   : IN     STD_LOGIC;                             --initiate transaction
    cpol     : IN     STD_LOGIC;                             --spi clock polarity
    cpha     : IN     STD_LOGIC;                             --spi clock phase
    cont     : IN     STD_LOGIC;                             --continuous mode command
    clk_div  : IN     INTEGER;                               --system clock cycles per 1/2 period of sclk
    addr     : IN     INTEGER;                               --address of slave
    tx_data  : IN     STD_LOGIC_VECTOR(d_width-1 DOWNTO 0);  --data to transmit
    miso     : IN     STD_LOGIC;                             --master in, slave out
    sclk     : INOUT  STD_LOGIC;                             --spi clock
    ss_n     : INOUT  STD_LOGIC_VECTOR(slaves-1 DOWNTO 0);   --slave select
    mosi     : OUT    STD_LOGIC;                             --master out, slave in
    busy     : OUT    STD_LOGIC;                             --busy / data ready signal
    rx_data  : OUT    STD_LOGIC_VECTOR(d_width-1 DOWNTO 0)); --data received
END spi_master;

ARCHITECTURE logic OF spi_master IS
  TYPE machine IS(ready, execute);                           --state machine data type
  SIGNAL state        : machine;                             --current state
  SIGNAL slave        : INTEGER;                             --slave selected for current transaction
  SIGNAL clk_ratio    : INTEGER;                             --current clk_div
  SIGNAL count        : INTEGER;                             --counter to trigger sclk from system clock
  SIGNAL clk_toggles  : INTEGER RANGE 0 TO d_width*2 + 1;    --count spi clock toggles
  SIGNAL assert_data  : STD_LOGIC;                           --'1' is tx sclk toggle, '0' is rx sclk toggle
  SIGNAL continue     : STD_LOGIC;                           --flag to continue transaction
  SIGNAL rx_buffer    : STD_LOGIC_VECTOR(d_width-1 DOWNTO 0); --receive data buffer
  SIGNAL tx_buffer    : STD_LOGIC_VECTOR(d_width-1 DOWNTO 0); --transmit data buffer
  SIGNAL last_bit_rx  : INTEGER RANGE 0 TO d_width*2;        --last rx data bit location
BEGIN
  PROCESS(clock, reset_n)
  BEGIN

    IF(reset_n = '0') THEN          --reset system
      busy <= '1';                  --set busy signal
      ss_n <= (OTHERS => '1');      --deassert all slave select lines
      mosi <= 'Z';                  --set master out to high impedance
      rx_data <= (OTHERS => '0');   --clear receive data port
      state <= ready;               --go to ready state when reset is exited

    ELSIF(clock'EVENT AND clock = '1') THEN
      CASE state IS                 --state machine

        WHEN ready =>
          busy <= '0';              --clock out not busy signal
          ss_n <= (OTHERS => '1');  --set all slave select outputs high
          mosi <= 'Z';              --set mosi output high impedance
          continue <= '0';          --clear continue flag

          --user input to initiate transaction
          IF(enable = '1') THEN
            busy <= '1';            --set busy signal
```

```
      IF(addr < slaves) THEN   --check for valid slave address
        slave <= addr;         --clock in current slave selection if valid
      ELSE
        slave <= 0;            --set to first slave if not valid
      END IF;
      IF(clk_div = 0) THEN     --check for valid spi speed
        clk_ratio <= 1;        --set to maximum speed if zero
        count <= 1;            --initiate system-to-spi clock counter
      ELSE
        clk_ratio <= clk_div;  --set to input selection if valid
        count <= clk_div;      --initiate system-to-spi clock counter
      END IF;
      sclk <= cpol;            --set spi clock polarity
      assert_data <= NOT cpha; --set spi clock phase
      tx_buffer <= tx_data;    --clock in data for transmit into buffer
      clk_toggles <= 0;        --initiate clock toggle counter
      last_bit_rx <= d_width*2 + conv_integer(cpha) - 1; --set last rx data bit
      state <= execute;        --proceed to execute state
    ELSE
      state <= ready;          --remain in ready state
    END IF;

  WHEN execute =>
    busy <= '1';         --set busy signal
    ss_n(slave) <= '0'; --set proper slave select output

    --system clock to sclk ratio is met
    IF(count = clk_ratio) THEN
      count <= 1;                      --reset system-to-spi clock counter
      assert_data <= NOT assert_data; --switch transmit/receive indicator
      IF(clk_toggles = d_width*2 + 1) THEN
        clk_toggles <= 0;                --reset spi clock toggles counter
      ELSE
        clk_toggles <= clk_toggles + 1; --increment spi clock toggles counter
      END IF;

      --spi clock toggle needed
      IF(clk_toggles <= d_width*2 AND ss_n(slave) = '0') THEN
        sclk <= NOT sclk; --toggle spi clock
      END IF;

      --receive spi clock toggle
      IF(assert_data = '0' AND clk_toggles < last_bit_rx + 1 AND ss_n(slave) = '0') THEN
        rx_buffer <= rx_buffer(d_width-2 DOWNTO 0) & miso; --shift in received bit
      END IF;

      --transmit spi clock toggle
      IF(assert_data = '1' AND clk_toggles < last_bit_rx) THEN
        mosi <= tx_buffer(d_width-1);                      --clock out data bit
        tx_buffer <= tx_buffer(d_width-2 DOWNTO 0) & '0'; --shift data transmit buffer
      END IF;

      --last data receive, but continue
      IF(clk_toggles = last_bit_rx AND cont = '1') THEN
        tx_buffer <= tx_data;                       --reload transmit buffer
        clk_toggles <= last_bit_rx - d_width*2 + 1; --reset spi clock toggle counter
```

```
            continue <= '1';                          --set continue flag
          END IF;

          --normal end of transaction, but continue
          IF(continue = '1') THEN
            continue <= '0';      --clear continue flag
            busy <= '0';          --clock out signal that first receive data is ready
            rx_data <= rx_buffer; --clock out received data to output port
          END IF;

          --end of transaction
          IF((clk_toggles = d_width*2 + 1) AND cont = '0') THEN
            busy <= '0';             --clock out not busy signal
            ss_n <= (OTHERS => '1'); --set all slave selects high
            mosi <= 'Z';             --set mosi output high impedance
            rx_data <= rx_buffer;    --clock out received data to output port
            state <= ready;          --return to ready state
          ELSE                       --not end of transaction
            state <= execute;        --remain in execute state
          END IF;

        ELSE        --system clock to sclk ratio not met
          count <= count + 1; --increment counter
          state <= execute;   --remain in execute state
        END IF;

    END CASE;
  END IF;
END PROCESS;
END logic;
```

Fuente: elaboración propia.

C. Bloques VHDL implementados en la plataforma para *debugging*

Modulo Top

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    13:33:59 12/04/2014
-- Design Name:
-- Module Name:    TOP - Struct
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
library UNISIM;
use IEEE.STD_LOGIC_1164.ALL;
use UNISIM.vcomponents.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity TOP is
    Port ( CLOCK : in STD_LOGIC;
                        RESET : in STD_LOGIC;
                        SEND : in STD_LOGIC;

                        CLKP : out STD_LOGIC;
                        CLKM : out STD_LOGIC;

                        ADC_DATA : in  STD_LOGIC_VECTOR (14 downto 0);
           ADC_SINK : in  STD_LOGIC;
           RX : in  STD_LOGIC;
```

```
            TX : out  STD_LOGIC;
            SDATA : out  STD_LOGIC;
            SCK : out  STD_LOGIC;
            SS : out  STD_LOGIC_VECTOR (2 downto 0);
            LOADDAC : out  STD_LOGIC;
            PWM_OUT : out  STD_LOGIC;
            LED : out  STD_LOGIC_VECTOR (7 downto 0);
            DISP : out  STD_LOGIC_VECTOR (7 downto 0);
            AN : out  STD_LOGIC_VECTOR (3 downto 0));
end TOP;

architecture Struct of TOP is

SIGNAL ADC_DAT : STD_LOGIC_VECTOR(14 DOWNTO 0);
SIGNAL ADCSINK, ADCSINK_SIGNAL : STD_LOGIC;
SIGNAL CLKP_SIGNAL, CLKP_SIGNAL2, CLKM_SIGNAL, CLKM_SIGNAL2 : STD_LOGIC;
SIGNAL CLK, RST : STD_LOGIC;

-- DECLARANDO COMPONENTES
-- UART RX
COMPONENT UART_RX
        PORT(
                CLK : IN std_logic;
                RST : IN std_logic;
                RX : IN std_logic;
                DATA : OUT std_logic_vector(7 downto 0);
                BUSY : OUT std_logic;
                READY : OUT std_logic;
                ERROR : OUT std_logic
                );
        END COMPONENT;

TYPE UARTRx is
RECORD
                DATA : std_logic_vector(7 downto 0);
                BUSY : std_logic;
                READY : std_logic;
                ERROR : std_logic;
END RECORD;

SIGNAL URX : UARTRx;


-- DESERIALIZER
COMPONENT UART2REG
        PORT(
                CLK : IN std_logic;
                RST : IN std_logic;
                DATA : IN std_logic_vector(7 downto 0);
                READY : IN std_logic;
                BUSY : IN std_logic;
                OUTPUT : OUT std_logic_vector(23 downto 0);
                DONE : OUT std_logic
                );
        END COMPONENT;
```

```
TYPE U2R_RECORD IS
RECORD
        OUTPUT : std_logic_vector(23 downto 0);
        DONE : std_logic;
END RECORD;

SIGNAL U2R : U2R_RECORD;



component CLK_GEN
port
 (-- Clock in ports
  CLOCK             : in      std_logic;
  -- Clock out ports
  CLK            : out     std_logic;
  CLKP            : out     std_logic;
  CLKM            : out     std_logic;
  -- Status and control signals
  RST               : in      std_logic
 );
end component;

        COMPONENT REGISTER_DRIVER
        PORT(
                CLK : IN std_logic;
                RST : IN std_logic;
                SEND_BUTTON : IN std_logic;
                REG_INPUT : IN std_logic_vector(23 downto 0);
                REG_DONE : IN std_logic;
                SPI_DATA_SET : OUT std_logic_vector(15 downto 0);
                SPI_SEND : OUT std_logic;
                FLANK : OUT std_logic;
                DEVICE : OUT INTEGER;
                LDAC : OUT std_logic;
                SS_Data : OUT std_logic_vector(15 downto 0);
                DEN : OUT std_logic_vector(3 downto 0);
                BLINK : OUT std_logic_vector(3 downto 0);
                DOT : OUT std_logic_vector(3 downto 0);
                VALUE : OUT std_logic_vector(15 downto 0);
                CLKDIV : OUT INTEGER;
                PWM_SEND : OUT std_logic;
                THREASHOLD : OUT std_logic_vector(13 downto 0);
                LED : OUT std_logic_vector(7 downto 0)
                );
        END COMPONENT;
-- Seven segments display
        COMPONENT SSDisplay
        PORT(
                CLK : IN std_logic;
                RST : IN std_logic;
                Data : IN std_logic_vector(15 downto 0);
                DEN : IN std_logic_vector(3 downto 0);
                BLINK : IN std_logic_vector(3 downto 0);
                DOT : IN std_logic_vector(3 downto 0);
                SS : OUT std_logic_vector(7 downto 0);
```

```
                En : OUT std_logic_vector(3 downto 0)
                );
        END COMPONENT;

TYPE SevSeg_RECORD IS
RECORD
                Data : std_logic_vector(15 downto 0);
                DEN : std_logic_vector(3 downto 0);
                BLINK : std_logic_vector(3 downto 0);
                DOT : std_logic_vector(3 downto 0);
END RECORD;

SIGNAL DISPLAY : SevSeg_RECORD;

--SPI COMPONENT
        COMPONENT SPI_W_LD
        PORT(
                CLK : IN std_logic;
                RST : IN std_logic;
                DATA : IN std_logic_vector(15 downto 0);
                SEND : IN std_logic;
                FLANK : IN std_logic;
                DEVICE : IN INTEGER;
                LDAC : IN std_logic;
                RLED : OUT std_logic;
                SCK : OUT std_logic;
                SDATA : OUT std_logic;
                LOADDAC : OUT std_logic;
                SS : OUT std_logic_vector(2 downto 0)
                );
        END COMPONENT;

TYPE SPI_RECORD IS
RECORD
                DATA : std_logic_vector(15 downto 0);
                SEND : std_logic;
                FLANK : std_logic;
                DEVICE : INTEGER;
                LDAC : std_logic;
                RLED : std_logic;
                SCK : std_logic;
                SDATA : std_logic;
                LOADDAC : std_logic;
                SS : std_logic_vector(2 downto 0);
END RECORD;

SIGNAL SPI : SPI_RECORD;

--PWM CONTROL
        COMPONENT PWM
        PORT(
                VALUE : IN std_logic_vector(15 downto 0);
                CLKDIV : IN integer;
                SEND : IN std_logic;
                CLK : IN std_logic;
                RST : IN std_logic;
```

```
                VOUT : OUT std_logic
                );
        END COMPONENT;

TYPE PWM_RECORD IS
RECORD
                VALUE : std_logic_vector(15 downto 0);
                CLKDIV : integer;
                SEND : std_logic;
                VOUT : std_logic;
END RECORD;

SIGNAL PWMS : PWM_RECORD;

COMPONENT DATA_SINK
        PORT(
                SINK_CLK : IN std_logic;
                RST : IN std_logic;
                SYS_CLK : IN std_logic;
                DATA_IN : IN std_logic_vector(14 downto 0);
                THREASHOLD : IN std_logic_vector(13 downto 0);
                TX : OUT std_logic
                );
        END COMPONENT;

SIGNAL THR : STD_LOGIC_VECTOR(13 DOWNTO 0);
-- FIN DECLARACION

begin

RST<= RESET;

-- Definiendo los buffers de entrada y salida.

--Buffer del ADC y sus Overflow.
ADCbuff: for i in 0 to 14 GENERATE
begin
IBUF_inst : IBUF port map(
        I => ADC_DATA(i),
        O => ADC_DAT(i)
);
end generate ADCbuff;

--Buffer del ADC_SINK
ADCSinkBuf: IBUFG  port map(
        I => ADC_SINK,
        O => ADCSINK_SIGNAL
);

ADCSinkbuf2: BUFG PORT MAP(
        I=> ADCSINK_SIGNAL,
        O=> ADCSINK
        );


CLKPBUF : BUFG port map(
```

```
        I => CLKP_SIGNAL,
        O => CLKP_SIGNAL2
);

ODDR2CLKP : ODDR2
port map (
Q => CLKP, -- 1-bit output data
C0 => CLKP_SIGNAL2, -- 1-bit clock input
C1 => NOT(CLKP_SIGNAL2), -- 1-bit clock input
D0 => '1', -- 1-bit data input (associated with C0)
D1 => '0', -- 1-bit data input (associated with C1)
R => RST -- 1-bit reset input
);


CLKMBUF : BUFG port map(
        I => CLKM_SIGNAL,
        O => CLKM_SIGNAL2
);

ODDR2CLKM : ODDR2
port map (
Q => CLKM, -- 1-bit output data
C0 => CLKM_SIGNAL2, -- 1-bit clock input
C1 => NOT(CLKM_SIGNAL2), -- 1-bit clock input
D0 => '1', -- 1-bit data input (associated with C0)
D1 => '0', -- 1-bit data input (associated with C1)
R => RST -- 1-bit reset input
);


--INSTANTIATION OF COMPONENTS

clk_generator : CLK_GEN
  port map
   (-- Clock in ports
    CLOCK => CLOCK,
    -- Clock out ports
    CLK => CLK,
    CLKP => CLKP_SIGNAL,
    CLKM => CLKM_SIGNAL,
    -- Status and control signals
    RST  => RST);

        RxPort: UART_RX PORT MAP(
                CLK => CLK,
                RST => RST,
                RX => RX,
                DATA => URX.DATA,
                BUSY => URX.BUSY,
                READY => URX.READY,
                ERROR => OPEN
        );

        Deser: UART2REG PORT MAP(
                CLK => CLK,
```

```
                RST => RST,
                DATA => URX.DATA,
                READY => URX.READY,
                BUSY => URX.BUSY,
                OUTPUT => U2R.OUTPUT,
                DONE => U2R.DONE
        );

SPILD: SPI_W_LD PORT MAP(
                CLK => CLK,
                RST => RST,
                DATA => SPI.DATA,
                SEND => SPI.SEND,
                FLANK => SPI.FLANK,
                DEVICE => SPI.DEVICE,
                LDAC => SPI.LDAC,
                RLED => SPI.RLED,
                SCK => SCK,
                SDATA => SDATA,
                LOADDAC => LOADDAC,
                SS => SS
        );


SevSegDisplay: SSDisplay PORT MAP(
                CLK => CLK,
                RST => RST,
                Data => DISPLAY.DATA,
                DEN => display.den,
                BLINK => display.blink,
                DOT => display.dot,
                SS => DISP,
                En => AN
        );

PWM_dev: PWM PORT MAP(
                VALUE => PWMS.VALUE,
                CLKDIV => PWMS.CLKDIV,
                SEND => PWMS.SEND,
                CLK => CLK,
                RST => RST,
                VOUT => PWM_OUT
        );

ADC_ACQQ: DATA_SINK PORT MAP(
                SINK_CLK => ADCSINK,
                RST => RST,
                SYS_CLK => CLK,
                DATA_IN => ADC_DAT,
                THREASHOLD => THR,
                TX => TX
        );


DRIVER: REGISTER_DRIVER PORT MAP(
                CLK => CLK,
```

```
            RST => RST,
            SEND_BUTTON => SEND,
            REG_INPUT => U2R.OUTPUT,
            REG_DONE => U2R.DONE,
            SPI_DATA_SET => SPI.DATA,
            SPI_SEND => SPI.SEND,
            FLANK => SPI.FLANK,
            DEVICE => SPI.DEVICE,
            LDAC => SPI.LDAC,
            SS_Data => display.data,
            DEN => display.den,
            BLINK => display.blink,
            DOT => display.dot,
            VALUE => PWMS.VALUE,
            CLKDIV => PWMS.CLKDIV,
            PWM_SEND => PWMS.SEND,
            THREASHOLD => THR,
            LED => LED
    );

end Struct;
```

Modulo UART2REG

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    16:53:08 11/25/2014
-- Design Name:
-- Module Name:    UART2REG - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity UART2REG is
    Port ( CLK : in std_logic;
                        RST : in STD_LOGIC;
                        DATA : in  STD_LOGIC_VECTOR(7 downto 0);
           READY : in  STD_LOGIC;
           BUSY : in  STD_LOGIC;
           OUTPUT : out  STD_LOGIC_VECTOR (23 downto 0);
           DONE : out  STD_LOGIC);
end UART2REG;

architecture Behavioral of UART2REG is

SIGNAL DATA_BUFF : STD_LOGIC_VECTOR(23 DOWNTO 0);

TYPE STATES IS (IDLE, W0, B0, W1, B1, W2);
signal state : STATES;
begin

process(rst, clk)
begin
if rst='1' then
        state<=IDLE;
        DATA_BUFF<=(OTHERS=>'0');
        DONE<='1';

ELSIF rising_edge(clk) then
        CASE state is
                when idle =>
                        if ready='0' then
                                state<=W0;
                                DONE<='0';
                        else
                                state<=idle;
                        end if;
                when w0 =>
                        if ready='1' then
                                DATA_BUFF(23 downto 16)<=DATA;
                                state<=B0;
                        else
                                state<=w0;
                        end if;
                when b0 =>
                        if ready='0' then
                                state<=W1;
                        else
                                state<=b0;
```

```
                end if;
            when W1 =>
                if ready='1' then
                    DATA_BUFF(15 DOWNTO 8)<=DATA;
                    STATE<=B1;
                else
                    state<=w1;
                END IF;
            WHEN B1 =>
                IF READY='0' THEN
                    STATE<=W2;
                else
                    state<=B1;
                END IF;
            WHEN W2 =>
                IF READY='1' THEN
                    DATA_BUFF(7 DOWNTO 0)<=DATA;
                    STATE<=IDLE;
                    DONE<='1';
                else
                    STATE<=W2;
                END IF;
            END CASE;

OUTPUT<=DATA_BUFF;



END IF;
end process;

end Behavioral;
```

Modulo UART TX CTRL

```
----------------------------------------------------------------------------------
--      UART_TX_CTRL.vhd -- UART Data Transfer Component
----------------------------------------------------------------------------------
-- Author: Sam Bobrowicz
--         Copyright 2011 Digilent, Inc.
----------------------------------------------------------------------------------
--
----------------------------------------------------------------------------------
--      This component may be used to transfer data over a UART device. It will
-- serialize a byte of data and transmit it over a TXD line. The serialized
-- data has the following characteristics:
--          *9600 Baud Rate
--          *8 data bits, LSB first
--          *1 stop bit
--          *no parity
--
-- Port Descriptions:
--
```

```
--     SEND - Used to trigger a send operation. The upper layer logic should
--            set this signal high for a single clock cycle to trigger a
--            send. When this signal is set high DATA must be valid . Should
--            not be asserted unless READY is high.
--     DATA - The parallel data to be sent. Must be valid the clock cycle
--            that SEND has gone high.
--      CLK - A 100 MHz clock is expected
--    READY - This signal goes low once a send operation has begun and
--            remains low until it has completed and the module is ready to
--            send another byte.
-- UART_TX - This signal should be routed to the appropriate TX pin of the
--            external UART device.
--
----------------------------------------------------------------------------
--
----------------------------------------------------------------------------
-- Revision History:
--  08/08/2011(SamB): Created using Xilinx Tools 13.2
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity UART_TX_CTRL is
    Port ( SEND : in  STD_LOGIC;
           DATA : in  STD_LOGIC_VECTOR (7 downto 0);
           CLK : in  STD_LOGIC;
           READY : out  STD_LOGIC;
           UART_TX : out  STD_LOGIC);
end UART_TX_CTRL;

architecture Behavioral of UART_TX_CTRL is

type TX_STATE_TYPE is (RDY, LOAD_BIT, SEND_BIT);

--constant BIT_TMR_MAX : std_logic_vector(13 downto 0) := "10100010110000"; --10416 = (round(100MHz / 9600)) - 1
constant BIT_TMR_MAX : std_logic_vector(13 downto 0) := "00001101100011"; --10416 = (round(100MHz / 115200)) - 1
constant BIT_INDEX_MAX : natural := 10;

--Counter that keeps track of the number of clock cycles the current bit has been held stable over the
--UART TX line. It is used to signal when the ne
signal bitTmr : std_logic_vector(13 downto 0) := (others => '0');

--combinatorial logic that goes high when bitTmr has counted to the proper value to ensure
--a 9600 baud rate
signal bitDone : std_logic;

--Contains the index of the next bit in txData that needs to be transferred
signal bitIndex : natural;

--a register that holds the current data being sent over the UART TX line
signal txBit : std_logic := '1';

--A register that contains the whole data packet to be sent, including start and stop bits.
signal txData : std_logic_vector(9 downto 0);
```

```
signal txState : TX_STATE_TYPE := RDY;

begin

--Next state logic
next_txState_process : process (CLK)
begin
        if (rising_edge(CLK)) then
                case txState is
                when RDY =>
                        if (SEND = '1') then
                                txState <= LOAD_BIT;
                        end if;
                when LOAD_BIT =>
                        txState <= SEND_BIT;
                when SEND_BIT =>
                        if (bitDone = '1') then
                                if (bitIndex = BIT_INDEX_MAX) then
                                        txState <= RDY;
                                else
                                        txState <= LOAD_BIT;
                                end if;
                        end if;
                when others=> --should never be reached
                        txState <= RDY;
                end case;
        end if;
end process;

bit_timing_process : process (CLK)
begin
        if (rising_edge(CLK)) then
                if (txState = RDY) then
                        bitTmr <= (others => '0');
                else
                        if (bitDone = '1') then
                                bitTmr <= (others => '0');
                        else
                                bitTmr <= bitTmr + 1;
                        end if;
                end if;
        end if;
end process;

bitDone <= '1' when (bitTmr = BIT_TMR_MAX) else
                                '0';

bit_counting_process : process (CLK)
begin
        if (rising_edge(CLK)) then
                if (txState = RDY) then
                        bitIndex <= 0;
                elsif (txState = LOAD_BIT) then
                        bitIndex <= bitIndex + 1;
                end if;
        end if;
```

```
end process;

tx_data_latch_process : process (CLK)
begin
        if (rising_edge(CLK)) then
                if (SEND = '1') then
                        txData <= '1' & DATA & '0';
                end if;
        end if;
end process;

tx_bit_process : process (CLK)
begin
        if (rising_edge(CLK)) then
                if (txState = RDY) then
                        txBit <= '1';
                elsif (txState = LOAD_BIT) then
                        txBit <= txData(bitIndex);
                end if;
        end if;
end process;

UART_TX <= txBit;
READY <= '1' when (txState = RDY) else
                        '0';

end Behavioral;
```

Modulo UART RX

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:     17:52:15 11/09/2014
-- Design Name:
-- Module Name:     UART_RX - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
```

```
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity UART_RX is
        GENERIC( baud_rate : integer := 115200;
                                --parity_bit : integer range 0 to 1:=0;
                                stop_bits : integer range 0 to 3 := 1);
    Port ( CLK : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           RX : in  STD_LOGIC;
           DATA : out  STD_LOGIC_VECTOR (7 downto 0);
           BUSY : out  STD_LOGIC;
           READY : out  STD_LOGIC;
                        ERROR : out STD_LOGIC);
end UART_RX;

architecture Behavioral of UART_RX is

--constant byte_number : integer := (9+stop_bits+parity_bit);
constant byte_number : integer := (9+stop_bits);
constant clk_div : integer := 100000000/baud_rate;

signal byte_count : integer range 0 to byte_number;
signal clk_count : integer range 0 to clk_div-1;
signal data_signal : std_logic_vector(7 downto 0);
signal busy_signal, ready_signal, parity : std_logic;

type states is (start, receive, finish, idle);
signal state : states;


begin

process(clk, rst, rx)
begin
if rst='1' then
        clk_count<=0;
        byte_count<=byte_number;
        data_signal<= (others=>'0');
        busy_signal <='0';
        ready_signal <='1';
        parity <='0';
        state<=idle;
elsif rising_edge(clk) then
        case state is
                when idle =>
                        byte_count<=0;
                        if rx='0' then
                                state<=start;
                                clk_count<=1;
                                busy_signal <='1';
                                ready_signal <='0';
```

```
                else
                        state<=idle;
                        clk_count<=0;
                        busy_signal<='0';
                end if;
        when start=>
                byte_count<=0;
                if clk_count= clk_div-1 then
                        state<=receive;
                        byte_count<=1;
                        clk_count<=0;
                else
                        state<=start;
                        clk_count<=clk_count+1;
                end if;
        when receive =>
                        if byte_count<9 then
                                if clk_count = clk_div/2 then
                                        data_signal(byte_count-1)<=rx;
                                        parity<=parity xor rx;
                                        clk_count<=clk_count+1;
                                elsif clk_count=clk_div-1 then
                                        byte_count<=byte_count+1;
                                        clk_count<=0;
                                else
                                        clk_count<=clk_count+1;
                                end if;
                        else
                                state<=finish;
                                Ready_signal<='1';
                                clk_count<=1;
                        end if;
        when finish=>
                if byte_count=byte_number then
                        state<=idle;
                elsif clk_count=clk_div-1 then
                        byte_count<=byte_count+1;
                        clk_count<=0;
                elsif clk_count=clk_div/2 then
                        busy_signal<='0';
                        clk_count<=clk_count+1;
                else
                        clk_count<=clk_count+1;
                end if;
        when others =>
                state<=idle;
        end case;
end if;
end process;

busy<=busy_signal;
ready<=ready_signal;
data<=data_signal;
```

```
end Behavioral;
```

Modulo SSPack

```
--
--      Package File Template
--
--      Purpose: This package defines supplemental types, subtypes,
--               constants, and functions
--
--   To use any of the example code shown below, uncomment the lines and modify as necessary
--

library IEEE;
use IEEE.STD_LOGIC_1164.all;

package SSPack is

-- type <new_type> is
--  record
--    <type_name>        : std_logic_vector( 7 downto 0);
--    <type_name>        : std_logic;
-- end record;
--
subtype cuatro_bit  is std_logic_vector(3 downto 0);
subtype siete_bit       is std_logic_vector(6 downto 0);
-- Declare constants
--
-- constant <constant_name>             : time := <time_unit> ns;
-- constant <constant_name>             : integer := <value;
--
-- Declare functions and procedure
--
-- function <function_name>  (signal <signal_name> : in <type_declaration>) return <type_declaration>;
-- procedure <procedure_name> (<type_declaration><constant_name>        : in <type_declaration>);
--
FUNCTION bin2bcd (bin : cuatro_bit) RETURN siete_bit;
FUNCTION enable(den : integer range 0 to 3; en : cuatro_bit) return cuatro_bit;
end SSPack;

package body SSPack is


FUNCTION bin2bcd (bin : cuatro_bit) RETURN siete_bit IS
variable bcd : siete_bit := "0000000";
BEGIN
case  bin is
        when x"0"=> bcd :="1000000"; -- '0'
        when x"1"=> bcd :="1111001"; -- '1'
        when x"2"=> bcd :="0100100"; -- '2'
        when x"3"=> bcd :="0110000";   -- '3'
        when x"4"=> bcd :="0011001"; -- '4'
        when x"5"=> bcd :="0010010";   -- '5'
        when x"6"=> bcd :="0000010"; -- '6'
```

```
        when x"7"=> bcd :="1111000"; -- '7'
        when x"8"=> bcd :="0000000"; -- '8'
        when x"9"=> bcd :="0010000"; -- '9'
         --nothing is displayed when a number more than 9 is given as input.
        -- Distribution-------ABCDEFG---------
        when x"A"=> bcd :="0001000"; -- 'A'
        when x"B"=> bcd :="0000011"; -- 'B'
        when x"C"=> bcd :="1000110"; -- 'C'
        when x"D"=> bcd :="0100001"; -- 'D'
        when x"E"=> bcd :="0000110"; -- 'E'
        when x"F"=> bcd :="0001110"; -- 'F'
        when others=> bcd :="1111111";
end case;
return bcd;
END bin2bcd;

FUNCTION enable(den : integer range 0 to 3; en : cuatro_bit) return cuatro_bit is
variable ret :cuatro_bit :="0000";
begin
        case den is
                when 0 => ret:=en and "0001";
                when 1 => ret:=en and "0010";
                when 2 => ret:=en and "0100";
                when 3 => ret:=en and "1000";
        end case;
        return ret;
end enable;
---- Example 1
--  function <function_name>  (signal <signal_name> : in <type_declaration>  ) return <type_declaration> is
--    variable <variable_name>      : <type_declaration>;
--  begin
--    <variable_name> := <signal_name> xor <signal_name>;
--    return <variable_name>;
--  end <function_name>;

---- Example 2
--  function <function_name>  (signal <signal_name> : in <type_declaration>;
--                             signal <signal_name>   : in <type_declaration>  ) return <type_declaration> is
--  begin
--    if (<signal_name> = '1') then
--      return <signal_name>;
--    else
--      return 'Z';
--    end if;
--  end <function_name>;

---- Procedure Example
--  procedure <procedure_name>  (<type_declaration><constant_name>  : in <type_declaration>) is
--
--  begin
--
--  end <procedure_name>;

end SSPack;
```

Modulo SSDisplay

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    16:57:05 10/18/2014
-- Design Name:
-- Module Name:    SSDisplay - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.SSPack.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity SSDisplay is
        generic( clk_div : integer :=100000);
    Port ( CLK   : in std_logic;
                        RST : in std_logic;
                        Data : in  STD_LOGIC_VECTOR (15 downto 0);
           DEN : in  STD_LOGIC_VECTOR (3 downto 0);
           BLINK : in  STD_LOGIC_VECTOR (3 downto 0);
           DOT : in  STD_LOGIC_VECTOR (3 downto 0);
           SS : out  STD_LOGIC_VECTOR (7 downto 0);
           En : out  STD_LOGIC_VECTOR (3 downto 0));
end SSDisplay;

architecture Behavioral of SSDisplay is

type display is
        record
        value : std_logic_vector(3 downto 0);
        display : std_logic_vector(7 downto 0);
        enable  : std_logic_vector(3 downto 0);
        end record;
```

```
signal d1 : display;




constant rst_disp : display := (        value => (others => '0'),
                                                                    display => (others => '0'),
                                                                    enable  => (others => '0'));

type displays is array (3 downto 0) of display;

signal d : displays;
constant rst_displays : displays :=(others => rst_disp);

signal d_count : integer range 0 to 3;

signal blink_clk : std_logic_vector(3 downto 0);


begin

process(clk, rst)
variable count : integer range 0 to clk_div-1;
variable blink_count :integer range 0 to 100;
begin
if rst='1' then
        d_count<=0;
        count:=0;
        d<=rst_displays;
        blink_clk<="0000";

elsif rising_edge(clk) then
        if count=clk_div-1 then
                count:=0;
                if d_count<3 then
                        d_count<=d_count+1;
                else
                        d_count<=0;
                        blink_count:=blink_count+1;
                end if;
        if blink_count =100 then
                blink_clk<=not(blink_clk);
                blink_count:=0;
        end if;
        else
                count:=count+1;
        end if;
        d(d_count).value<=Data((3+4*d_count) downto (0+4*d_count));
        d(d_count).display<=not(Dot(d_count)) &  bin2bcd(Data((3+4*d_count) downto (0+4*d_count)));
        d(d_count).enable<=enable(d_count, DEN);
end if;
end process;

process(d, d_count, blink)
begin
```

```
SS<=d(d_count).display;
En<=not(d(d_count).enable) xor (enable(d_count, blink) and blink_clk);
end process;


end Behavioral;
```

Modulo REGISTER DRIVER

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    18:54:13 11/16/2014
-- Design Name:
-- Module Name:    REGISTER_DRIVER - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity REGISTER_DRIVER is
        Generic (Reg_length : integer:= 23; BUS_LENGTH :INTEGER := 15);
        Port (  CLK : IN STD_LOGIC;
                        RST : IN STD_LOGIC;
                        SEND_BUTTON : IN STD_LOGIC;
         --UART INPUT
                        REG_INPUT : IN STD_LOGIC_VECTOR (REG_LENGTH DOWNTO 0);
                        REG_DONE        : IN STD_LOGIC;
                        --SPI OUTPUTS
            SPI_DATA_SET : out  STD_LOGIC_VECTOR(BUS_LENGTH DOWNTO 0);
            SPI_SEND : out STD_LOGIC;
            FLANK : out  STD_LOGIC;
            DEVICE : out  INTEGER;
            LDAC : out STD_LOGIC;
```

```
                        --SEVEN SEGMENTS DISPLAY
                        SS_Data : out  STD_LOGIC_VECTOR (BUS_LENGTH downto 0);
            DEN : out STD_LOGIC_VECTOR (3 downto 0);
            BLINK : out  STD_LOGIC_VECTOR (3 downto 0);
            DOT : out  STD_LOGIC_VECTOR (3 downto 0);
                        --PWM
                        VALUE : out  STD_LOGIC_VECTOR (BUS_LENGTH downto 0);
                        CLKDIV : out integer;
                        PWM_SEND : out STD_LOGIC;
                        --THREASHOLD
                        THREASHOLD : out STD_LOGIC_VECTOR(13 DOWNTO 0);
                        LED : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)


                        );
end REGISTER_DRIVER;

architecture Behavioral of REGISTER_DRIVER is

--Defining register values
SIGNAL ADD : STD_LOGIC_VECTOR(7 DOWNTO 0);
signal GPR : STD_LOGIC_VECTOR(15 DOWNTO 0);
--SPI REGISTER ADDRESS
constant SPI_DATA : STD_LOGIC_VECTOR(7 DOWNTO 0) := x"10"; --TWO BYTE WAITING

constant SPI_INFO : STD_LOGIC_VECTOR(7 DOWNTO 0) := x"11"; --ONE BYTE WAITING
signal SPI_CONF : STD_LOGIC_VECTOR(7 DOWNTO 0);
--REGISTER CONFIGURATION
--7            FLANK
--6            LOAD DAC
--5            AUTO SEND, 1 AUTO, 0 MANUAL
--4,3   -NI
--2-0   DEVICE

TYPE SPI_DEVICE IS
RECORD
            DATA : STD_LOGIC_VECTOR(15 DOWNTO 0);
            SEND : STD_LOGIC;
            FLANK : STD_LOGIC;
            DEVICE : INTEGER;
            LDAC : STD_LOGIC;
END RECORD;
signal SPI : SPI_DEVICE;

--PWM REGISTERS
CONSTANT PWM_DATA : STD_LOGIC_VECTOR (7 DOWNTO 0) := x"20"; --TWO BYTE WAITING
CONSTANT PWM_REG          : STD_LOGIC_VECTOR (7 DOWNTO 0) := x"21"; --ONE BYTE WAITING
signal PWM_CONF : STD_LOGIC_VECTOR (7 DOWNTO 0);
--7 on-off bit
--2-0 clk_div

TYPE PWM_DEVICE IS
RECORD
                        VALUE : STD_LOGIC_VECTOR (15 downto 0);
                        CLKDIV : integer;
                        SEND : STD_LOGIC;
```

```
END RECORD;

SIGNAL PWM : PWM_DEVICE;

--SET THREASHOLD
CONSTANT THR_REG : STD_LOGIC_VECTOR(7 DOWNTO 0):=x"30"; --TWO BYTE WAITING



SIGNAL ADDRESS, RDATA0, RDATA1 : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL BYTE_WAITING : INTEGER RANGE 0 TO 2;

SIGNAL THR : STD_LOGIC_VECTOR(13 DOWNTO 0);

begin

ADD<= REG_INPUT(23 DOWNTO 16) WHEN REG_DONE='1' ELSE (OTHERS=>'Z');
GPR<=REG_INPUT(15 DOWNTO 0) WHEN REG_DONE='1' ELSE (OTHERS=>'Z');


--Configuring SPI in the output
process(ADD, RST, clk)
begin
if RST='1' THEN
        SPI.DATA<=(OTHERS=>'0');
        SPI_CONF<=(OTHERS=>'0');
        PWM.VALUE<=(OTHERS=>'0');
        PWM_CONF<=(OTHERS=>'0');
        THR<=(OTHERS=>'1');
ELSIF RISING_EDGE(CLK) THEN
case ADD is
        when SPI_DATA =>
                SPI.DATA<=GPR;
        WHEN SPI_INFO =>
                SPI_CONF<=GPR(15 DOWNTO 8);
        WHEN PWM_DATA =>
                PWM.VALUE<=GPR;
        WHEN PWM_REG =>
                PWM_CONF<=GPR(15 DOWNTO 8);
        WHEN THR_REG =>
                THR<=GPR(13 DOWNTO 0);
        WHEN OTHERS =>

        END CASE;
END IF;
end process;

--SPI.DATA<=GPR WHEN ADD=x"10" else
--                        (OTHERs=>'0') WHEN RST='1' ELSE
--                         SPI.DATA;
--
--SPI_CONF<=GPR(15 DOWNTO 8) WHEN ADD=X"11" else
--                        (OTHERs=>'0') WHEN RST='1' ELSE
--                         SPI_CONF;

SPI.FLANK<=SPI_CONF(7);
```

```
SPI.LDAC<=SPI_CONF(6);
SPI.DEVICE<=     0 WHEN SPI_CONF(2 DOWNTO 0) ="000" ELSE
                                          1 WHEN SPI_CONF(2 DOWNTO 0) ="001" ELSE
                                          2 WHEN SPI_CONF(2 DOWNTO 0) ="010" ELSE
                                          0;


SPI.SEND<=SPI_CONF(5) OR SEND_BUTTON;


--Configuring PWM

--PWM.VALUE<=GPR WHEN ADD=x"20" else
--                                  (others=>'0') when rst='1' else
--                                   PWM.VALUE;
--
--PWM_CONF<=GPR(15 DOWNTO 8) WHEN ADD=x"22" else
--                           (others=>'0') when rst='1' else
--                           pwm_conf;
PWM.CLKDIV<=     1 when PWM_CONF(2 DOWNTO 0)="000" ELSE
                                          2 when PWM_CONF(2 DOWNTO 0)="001" ELSE
                                          3 when PWM_CONF(2 DOWNTO 0)="010" ELSE
                                          4 when PWM_CONF(2 DOWNTO 0)="011" ELSE
                                          5 when PWM_CONF(2 DOWNTO 0)="100" ELSE
                                          6 when PWM_CONF(2 DOWNTO 0)="101" ELSE
                                          7 when PWM_CONF(2 DOWNTO 0)="110" ELSE
                                          8 when PWM_CONF(2 DOWNTO 0)="111" ELSE
                                          1;
PWM.SEND<=PWM_CONF(7);

--Configuring Threashold
--THR<=GPR(13 DOWNTO 0) when add=x"30" else
--                                  (others=>'0') when rst='1' else
--                                   THR;

THREASHOLD<=THR;
--setting the outputs

SPI_DATA_SET <=SPI.DATA;
SPI_SEND <=SPI.SEND;
FLANK <= SPI.FLANK;
DEVICE <= SPI.DEVICE;
LDAC <= SPI.LDAC;

SS_Data <= GPR;
DEN <= (OTHERS=>'1');
BLINK <= (OTHERS=>'0');
DOT <= "0100";

LED<= ADD;
--PWM
VALUE <= PWM.VALUE;
CLKDIV <= PWM.CLKDIV;
PWM_SEND <= PWM.SEND;

end Behavioral;
```

Modulo PWM

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    22:01:20 07/17/2014
-- Design Name:
-- Module Name:    PWM - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity PWM is
        Generic(BitNum : integer := 15);
    Port ( VALUE : in  STD_LOGIC_VECTOR (BitNum downto 0);
                       CLKDIV : in integer;
                       SEND : in STD_LOGIC;
           CLK : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           VOUT : out  STD_LOGIC);
end PWM;

architecture Behavioral of PWM is

constant MAX_COUNT : STD_LOGIC_VECTOR(BitNum downto 0) := (OTHERS=>'1');
signal BUF_A : STD_LOGIC_VECTOR(BitNum downto 0) := (OTHERS=>'0');
signal BUF_B : STD_LOGIC_VECTOR(BitNum downto 0) := (OTHERS=>'0');
signal count : STD_LOGIC_VECTOR(BitNum downto 0) := (OTHERS=>'0');
signal out_sig: std_logic :='0';
signal cnt : integer;

begin
```

```
BUF_A<=MAX_COUNT;
BUF_B<=VALUE;
VOUT<=out_sig and SEND;

process(CLK, RST)

begin
if RST='1' then
        count<=(OTHERS=>'0');
        cnt <= 0;
elsif RISING_EDGE(CLK) THEN
        IF cnt=CLKDIV then
                cnt<=0;
                IF count=BUF_A then
                        count<=(others =>'0');
                ELSE
                        count<=count+1;
                END IF;
        ELSE
                cnt<=cnt+1;
        END IF;
END IF;
end process;

out_sig<= '0' WHEN RST='1' ELSE
                        '1' WHEN count<=BUF_B ELSE
                        '0';




end Behavioral;
```

Modulo DATA SINK

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:     14:48:57 11/13/2014
-- Design Name:
-- Module Name:     DATA_SINK - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
```

```
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity DATA_SINK is
    Port ( SINK_CLK : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           SYS_CLK : in  STD_LOGIC;
           DATA_IN : in  STD_LOGIC_VECTOR (14 downto 0);
           THREASHOLD : in  STD_LOGIC_VECTOR(13 downto 0);
           TX : out  STD_LOGIC);
end DATA_SINK;

architecture Behavioral of DATA_SINK is

type states is (RST_STATE,waiting, acquire, transfer);
signal state : states;

type fifo_reg is
record
        rst : STD_LOGIC;
    wr_clk : STD_LOGIC;
    rd_clk : STD_LOGIC;
    din : STD_LOGIC_VECTOR(14 DOWNTO 0);
    wr_en : STD_LOGIC;
    rd_en : STD_LOGIC;
    dout : STD_LOGIC_VECTOR(14 DOWNTO 0);
    full : STD_LOGIC;
    empty : STD_LOGIC;
end record;
signal rfifo : fifo_reg;
SIGNAL DATA_TMP : STD_LOGIC_VECTOR (15 DOWNTO 0);


COMPONENT FIFO
  PORT (
    rst : IN STD_LOGIC;
    wr_clk : IN STD_LOGIC;
    rd_clk : IN STD_LOGIC;
    din : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
    wr_en : IN STD_LOGIC;
    rd_en : IN STD_LOGIC;
    dout : OUT STD_LOGIC_VECTOR(14 DOWNTO 0);
    full : OUT STD_LOGIC;
```

```
    empty : OUT STD_LOGIC
  );
END COMPONENT;


COMPONENT UART_TX_CTRL
        PORT(
                SEND : IN std_logic;
                DATA : IN std_logic_vector(7 downto 0);
                CLK : IN std_logic;
                READY : OUT std_logic;
                UART_TX : OUT std_logic
                );
END COMPONENT;

TYPE UART_SIGNALS IS
RECORD
        SEND : STD_LOGIC;
        DATA : STD_LOGIC_VECTOR(7 DOWNTO 0);
        CLK  : STD_LOGIC;
        READY    : STD_LOGIC;
        TX       : STD_LOGIC;
END RECORD;

signal UART : UART_SIGNALS;

begin
FIFO_MEM : FIFO
  PORT MAP (
    rst => rfifo.rst,
    wr_clk => rfifo.wr_clk,
    rd_clk => rfifo.rd_clk,
    din => rfifo.din,
    wr_en => rfifo.wr_en,
    rd_en => rfifo.rd_en,
    dout => rfifo.dout,
    full => rfifo.full,
    empty => rfifo.empty
  );


        Inst_UART_TX_CTRL: UART_TX_CTRL PORT MAP(
                SEND => UART.SEND,
                DATA => UART.DATA,
                CLK => UART.CLK,
                READY => UART.READY,
                UART_TX => UART.TX
        );


process(sink_clk, rst, rfifo.full, rfifo.empty)
begin
if rst='1' then
        rfifo.wr_en<='0';
        rfifo.rd_en<='0';
        state<=RST_STATE;
```

```
elsif rising_edge(sink_clk) then
        CASE STATE IS
                when RST_STATE=>
                        IF rfifo.full='0' and rfifo.empty='1' then
                                state<=WAITING;
                        else
                                state<=RST_STATE;
                        END IF;
                WHEN WAITING =>
                        IF DATA_IN(13 DOWNTO 0)>THREASHOLD  THEN
                                STATE<=ACQUIRE;
                                rfifo.wr_en<='1';
                                rfifo.rd_en<='0';
                        ELSE
                                STATE<=WAITING;
                                rfifo.wr_en<='0';
                                rfifo.rd_en<='0';
                        END IF;
                WHEN ACQUIRE =>
                        if rfifo.full='1' then
                                rfifo.wr_en<='0';
                                rfifo.rd_en<='1';
                                STATE<=TRANSFER;
                        else
                                STATE<=ACQUIRE;
                        END IF;
                WHEN TRANSFER =>
                        IF rfifo.full='0' and rfifo.empty='1' then
                                rfifo.wr_en<='0';
                                rfifo.rd_en<='0';
                                STATE<=WAITING;
                        ELSE
                                STATE<=TRANSFER;
                        END IF;
                end case;
END IF;
end process;



UART.SEND<='1' WHEN STATE=TRANSFER ELSE '0';
DATA_TMP<=('1' & rfifo.dout); --WHEN rfifo.rd_clk = '1' else (others => '0');
UART.DATA<= DATA_TMP(15 DOWNTO 8) WHEN RFIFO.RD_CLK='1' ELSE
                                DATA_TMP(7 DOWNTO 0) WHEN RFIFO.RD_CLK='0' ELSE
                                (OTHERS =>'0');


transfer_clock:process(UART.READY, RST)
BEGIN
IF RST ='1' THEN
        rfifo.rd_clk<='0';
elsif UART.READY='1' THEN
        rfifo.rd_clk<=not(rfifo.rd_clk);
end if;
end process;
```

```
UART.CLK<=SYS_CLK;
rfifo.rst<=rst;
TX<=UART.TX;

rfifo.din<=DATA_IN;
rfifo.wr_clk<=sink_clk;



end Behavioral;
```

Modulo CLK MANAGER

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    21:02:54 11/02/2014
-- Design Name:
-- Module Name:    CLK_MANAGER - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity CLK_MANAGER is
    Port ( CLK : in  STD_LOGIC;
           CLK_BUFFER : out  STD_LOGIC;
           CLKP : out  STD_LOGIC;
           CLKM : out  STD_LOGIC;
           RST : in  STD_LOGIC);
end CLK_MANAGER;

architecture Behavioral of CLK_MANAGER is
--component CLOCKING
```

```
--port
-- (-- Clock in ports
--   CLK_IN1           : in     std_logic;
--   -- Clock out ports
--   CLK_OUT1          : out    std_logic;
--   CLK_OUT2          : out    std_logic;
--   CLK_OUT3          : out    std_logic;
--   -- Status and control signals
--   RESET             : in     std_logic
-- );
--end component;
SIGNAL CLK_SIGNAL : STD_LOGIC;
begin

CLK_SIGNAL<=CLK;
CLK_BUFFER<=CLK_SIGNAL;
CLKP<=CLK_SIGNAL;
CLKM<=NOT(CLK_SIGNAL);
--CLK_MANAGER : CLOCKING
--   port map
--    (-- Clock in ports
--     CLK_IN1 => CLK,
--     -- Clock out ports
--     CLK_OUT1 => CLK_BUFFER,
--     CLK_OUT2 => CLKP,
--     CLK_OUT3 => CLKM,
--     -- Status and control signals
--     RESET  => RST);
--
end Behavioral;
```

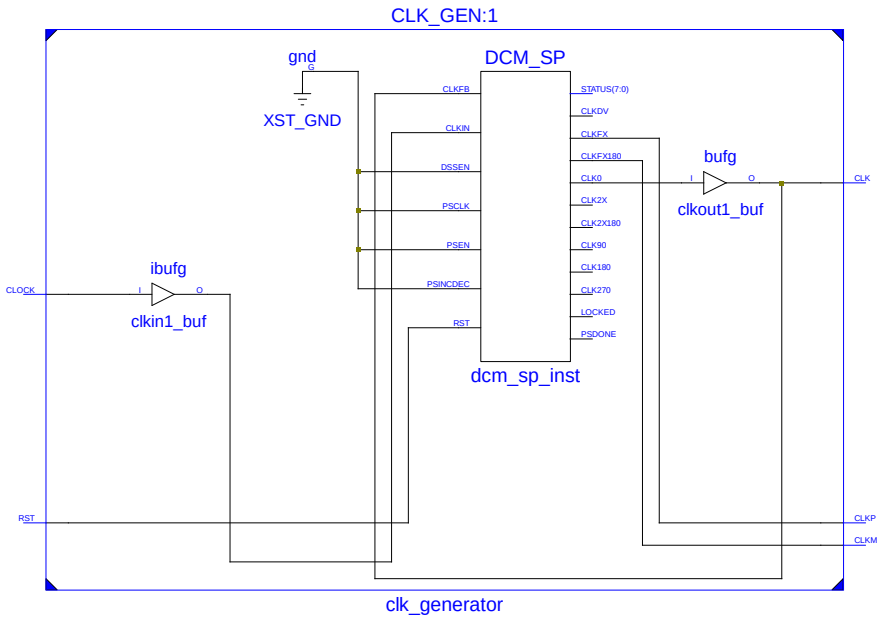
Fuente:

elaboración propia.

D. Bloques funcionales utilizados en el *debugger*

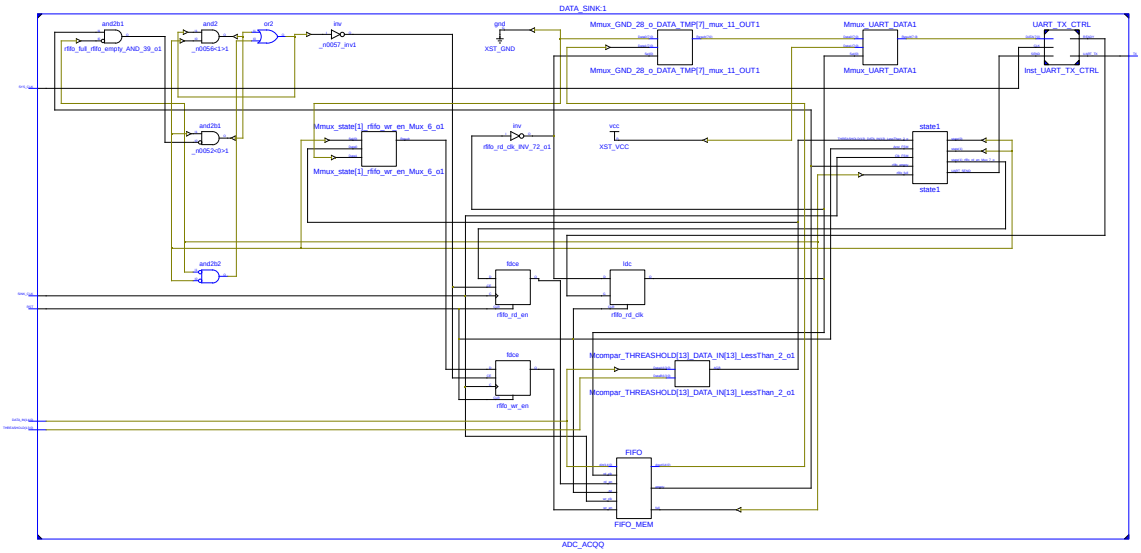
Los siguientes bloques funcionales fueron sintetizados utilizando el programa XILINX ISE Design Suite utilizando el código VHDL citado en la sección anterior.

Bloque funcional para la función generadora del reloj



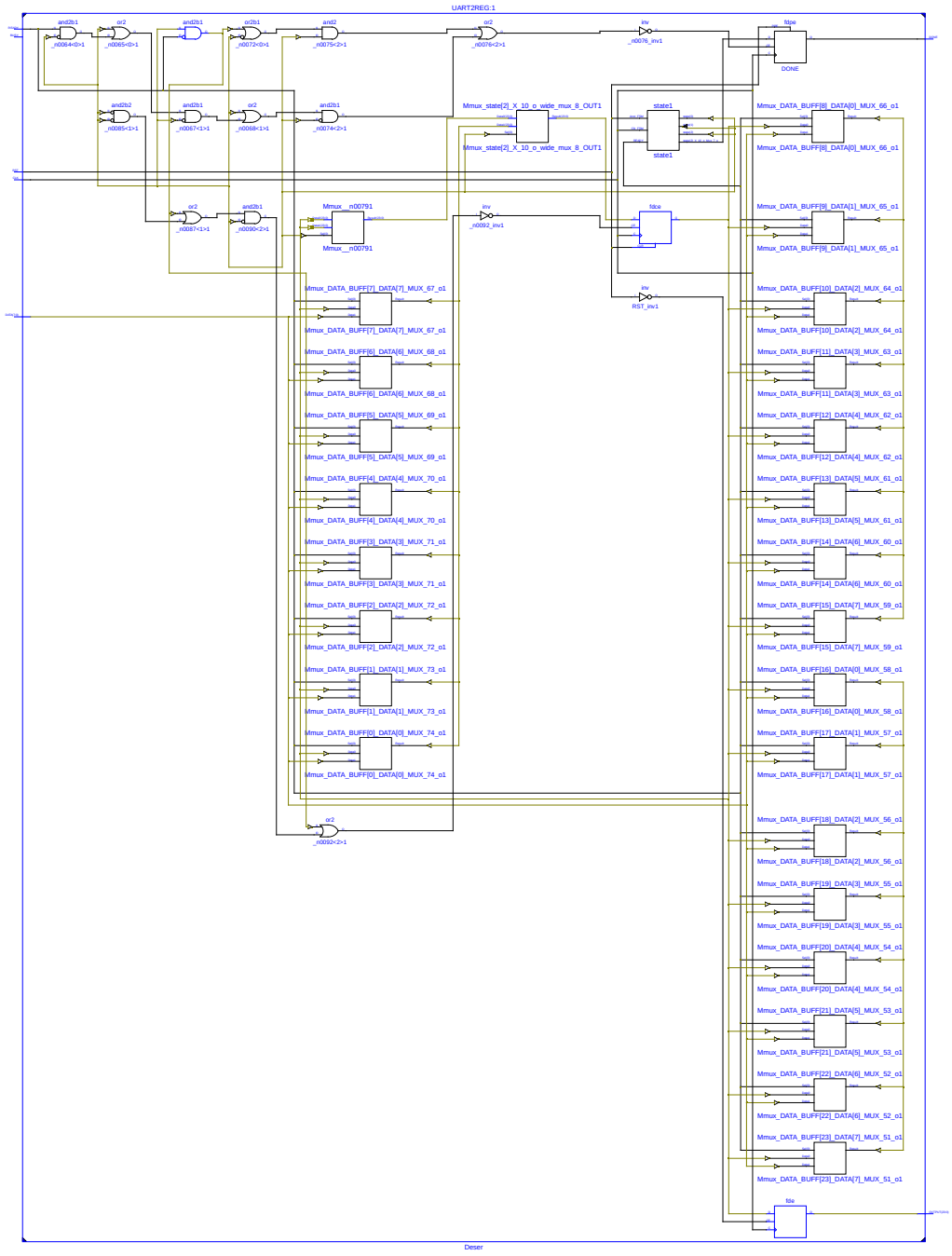
Fuente: elaboración propia.

Bloque funcional para la adquisición del ADC



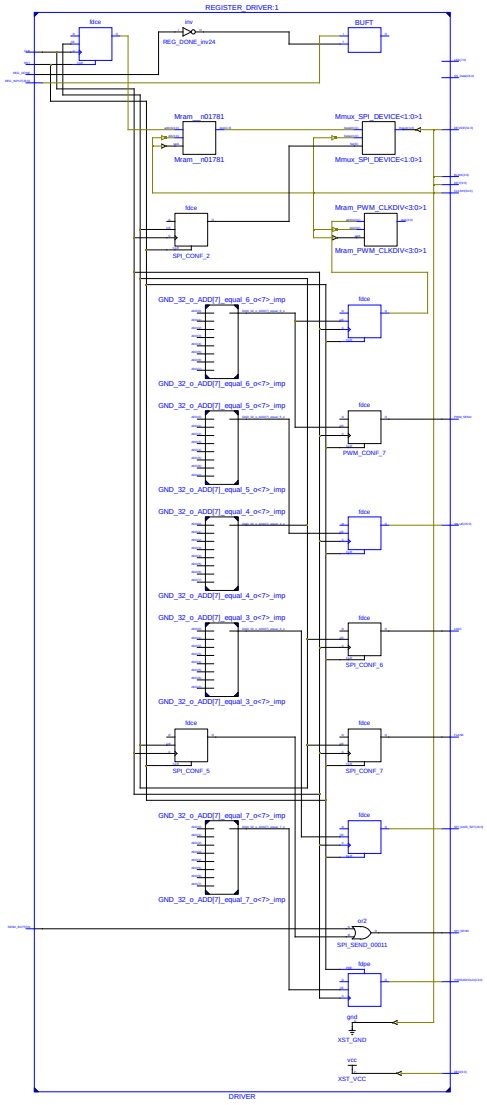
Fuente: elaboración propia.

Bloque funcional para la recepción serial



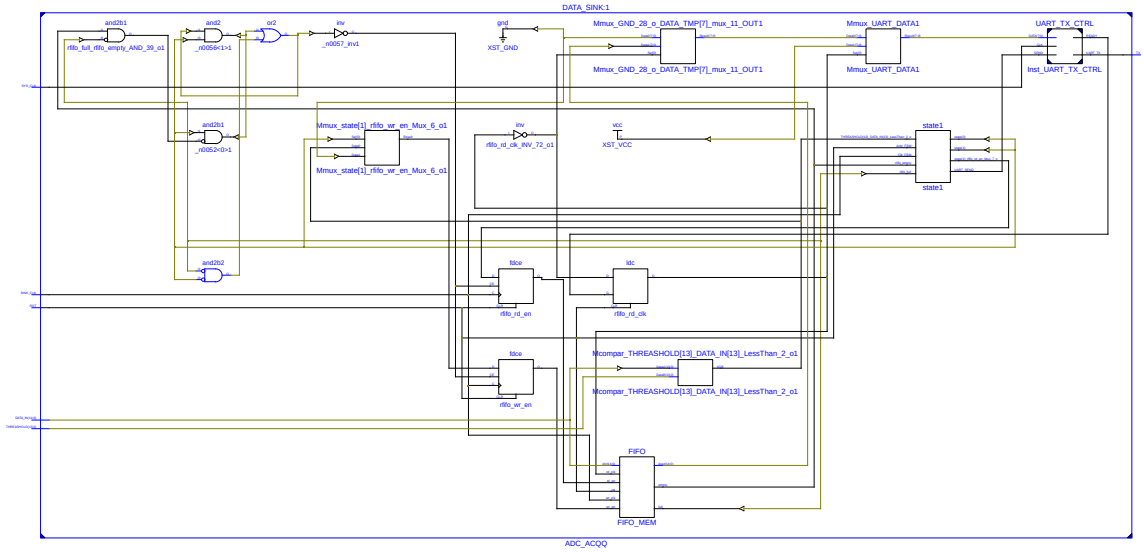
Fuente: elaboración propia.

Bloque funcional para el control de registros



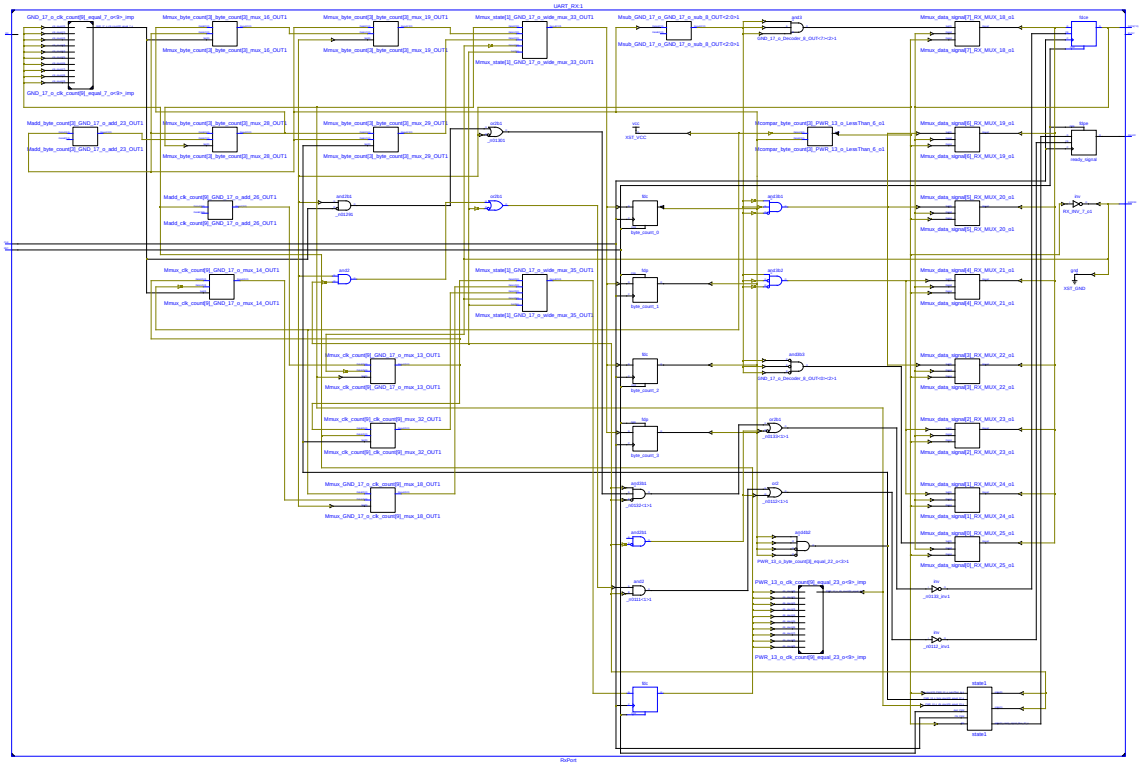
Fuente: elaboración propia.

Bloque funcional para generación del PWM



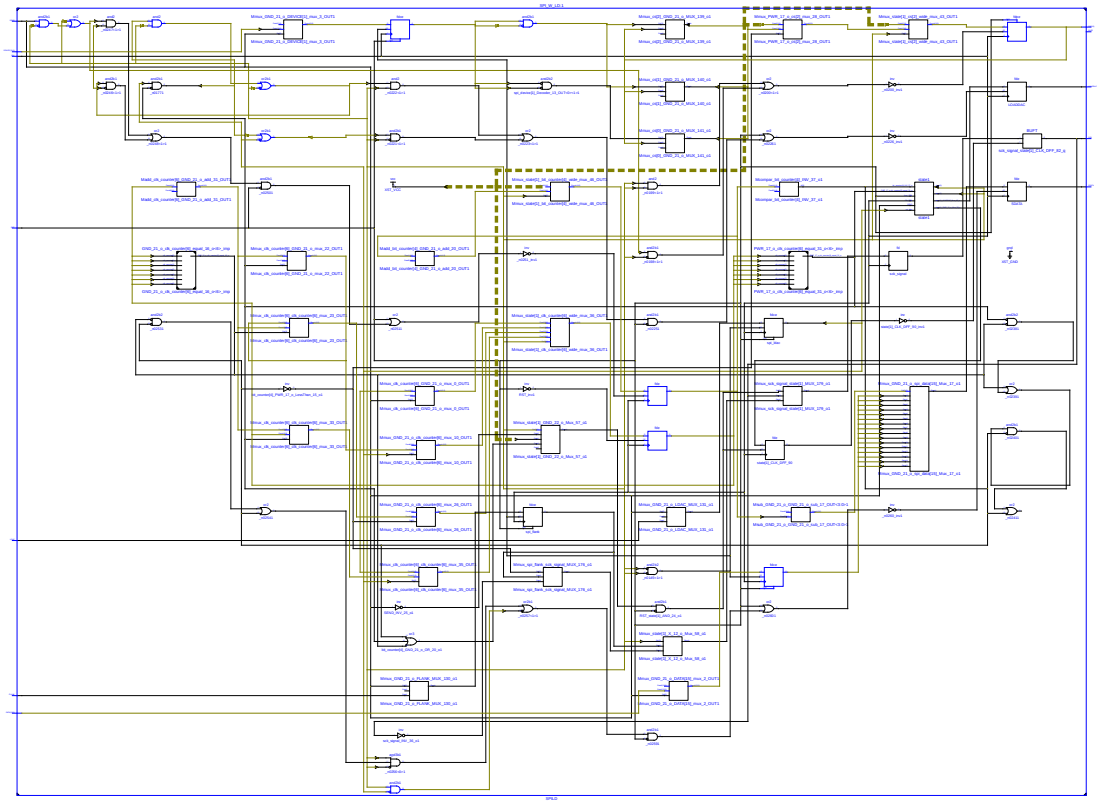
Fuente: elaboración propia.

Bloque funcional para transmisión serial



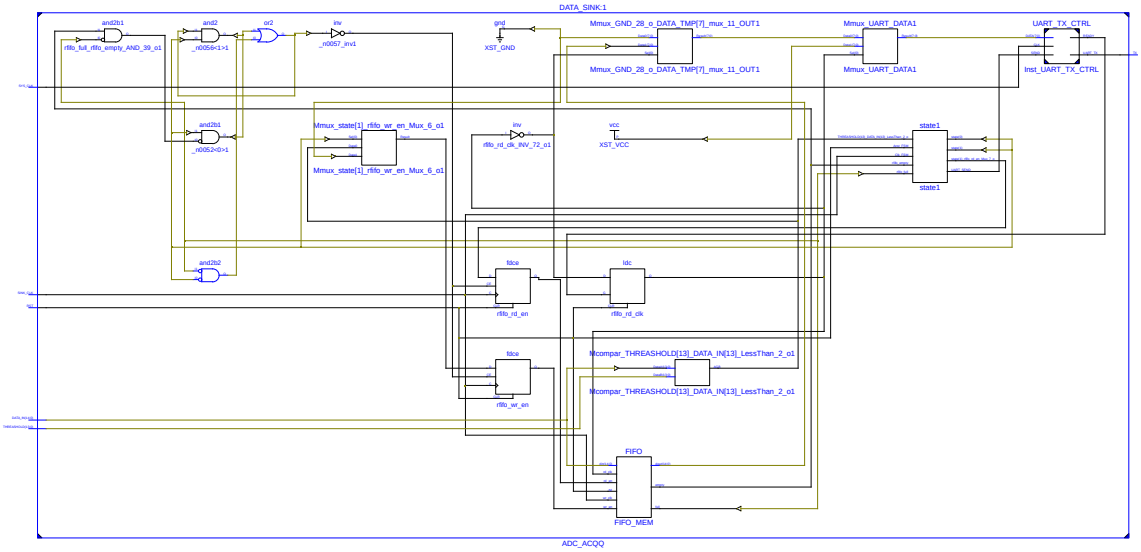
Fuente: elaboración propia.

Bloque funcional para comunicación SPI



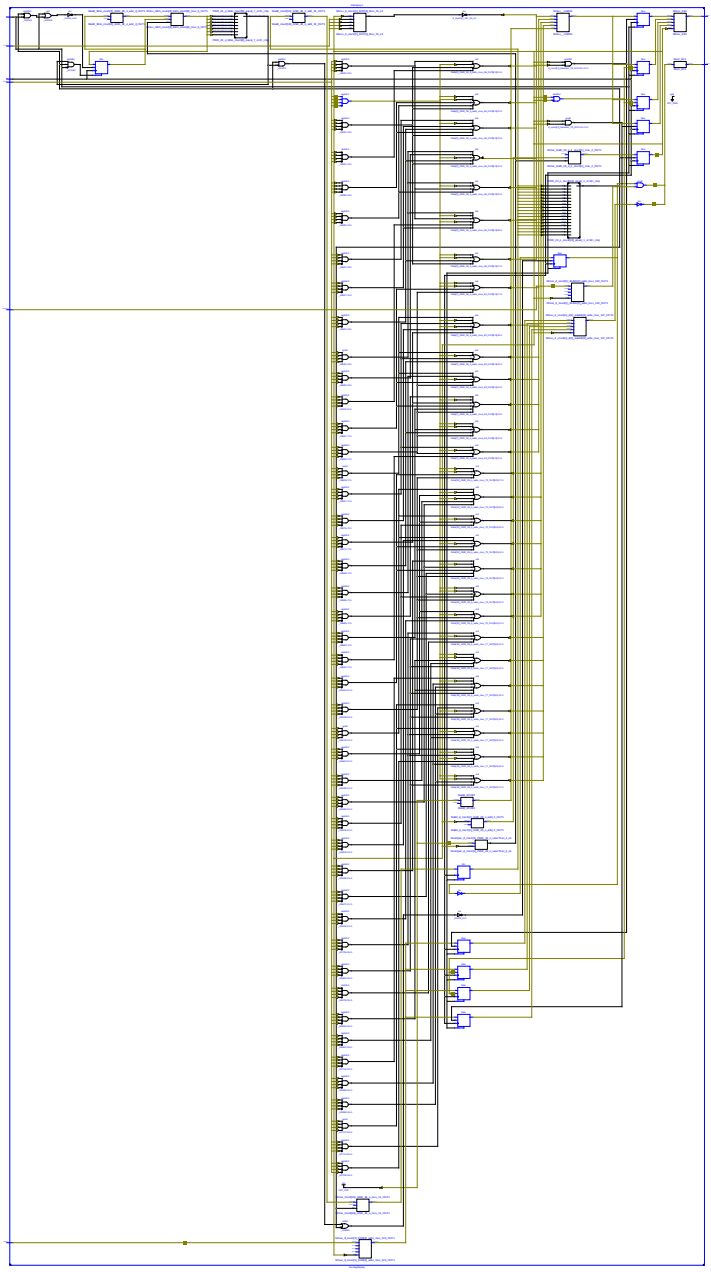
Fuente: elaboración propia.

Bloque funcional para generación del PWM



Fuente: elaboración propia.

Bloque funcional para retroalimentación visual



Fuente: elaboración propia.

E. Script de prueba ejecutado en Python 2.7

En el siguiente capitulo se detalla la librería desarrollada y el *script* de prueba utilizado para la adquisición de datos simulados por la estación de *debugging*. El *script* corre sobre python 2.7 utilizando los paquetes Matplotlib, PySerial, Struct los cuales deben estar instalados en el equipo y debe tener la librería uif.py en la misma carpeta.

Librería uif.py (*User Interface*)

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from Tkinter import *
import tkFileDialog
import tkMessageBox
import time
import ttk
#import pylab
import serial
import struct

class serIface:
    def startUART(self, puerto='COM7', baud=115200):
        self.puerto=puerto
        self.baud=baud
        self.puerto=serial.Serial(self.puerto, baudrate=self.baud)

    def sendADC(self, val):
        self.puerto.write('\x11\x00\x00')
        a=struct.pack('H', val)
        self.puerto.write('\x10'+a[1]+a[0])

    def sendSLC(self, val):
        self.puerto.write('\x11\x41\x00')
        a=struct.pack('H', val)
        self.puerto.write('\x10'+a[1]+a[0])

    def sendBLC(self, val):
        self.puerto.write('\x11\xc2\x00')
        a=struct.pack('H', val)
        self.puerto.write('\x10'+a[1]+a[0])
```

```
def changeVal(self, val):
     a=struct.pack('H', val)
     self.puerto.write('\x10'+a[1]+a[0])

def dinamicSPI(self, dev, on=True):
    devs=['ADC', 'SLCTRL', 'BLCTRL']
    if dev in devs:
        if on==True:
            var=''
            if dev=='ADC':
                var='\x20'
            elif dev=='SLCTRL':
                var='\x61'
            elif dev=='BLCTRL':
                var='\xe2'
            else:
                var=='\x00'
            self.puerto.write('\x11'+var+'\x00')
        else:
            self.puerto.write('\x11\x00\x00')

def PWM_STATE(self, on=True):
    if on==True:
        self.puerto.write('\x21\xf0\x00')
    else:
        self.puerto.write('\x21\x00\x00')

def PWM_SET(self, val):
     a=struct.pack('H', val)
     self.puerto.write('\x20'+a[1]+a[0])

def THR_SET(self, val):
     a=struct.pack('H', val)
     self.puerto.write('\x30'+a[1]+a[0])

def getData(self, num=2048):
    a=self.puerto.read(num)
    data=[]
    for val in range(0,len(a),2):
        data.append(struct.unpack('H', a[val]+a[val-1]))
    return data

def getData2(self, num=2048):
    a=self.puerto.read(num)
    data=[]
    for val in range(0,len(a),2):
        data.append(struct.unpack('H', a[val-1]+a[val])[0])
    return data

def getData3(self, num=2048):
    a=self.puerto.read(num)
    dat=0
    data=[]
    for val in range(0,len(a),2):
        dat=struct.unpack('H', a[val-1]+a[val])[0]
        data.append(dat%(2**14))
```

```
        return data

    def saveData(self, data):
        f=open('c:/Users/KBo/Desktop/result.csv','w')
        for d in data:
            f.write(str(d)+'\n')
        f.close()


    def plotData(self, data):
        pylab.plot(range(len(data)), data)
```

Script de control

```
from uif import * #Cargando la libreria uif


raw_input('Resetee la FPGA por favor')
ser=serIface()
ser.startUART('COM8') #abriendo la interfaz serial en el puerto 8

ser.dinamicSPI('BLCTRL') #Colocando en 0 (2000 representado rango a rango) el valor del BLCTRL
ser.changeVal(2000)

ser.dinamicSPI('BLCTRL',on=False) #Apagando el BLCTRL

ser.dinamicSPI('SLCTRL') #Polarizando el PMT
ser.changeVal(33728)
ser.dinamicSPI('SLCTRL', on=False)

while ser.puerto.inWaiting()!=0: #Limpiando cualquier entrada del modulo serial
    ser.puerto.flushInput()

while ser.puerto.inWaiting()!=0:
    ser.puerto.flushInput()

ser.THR_SET(33000) #Colocando un valor del Threashold


print ser.puerto.inWaiting()
raw_input('Inicio') #Esperando para encender el PWM
ser.PWM_SET(20) #Seteando el valor de pwm
ser.PWM_STATE()

time.sleep(0.01)

data=ser.getData3() #Espera datos de la interfaz uart
ser.saveData(data) #almacena los datos en un archivo para su posterior analisis.
```

Fuente: elaboración propia.