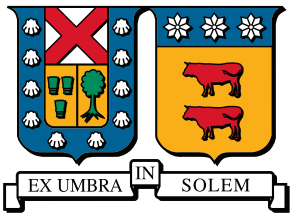
2020-12

DISEÑO DE UNA PLACA DE CONTROL PARA EL MÓDULO DE UN TRANSFORMADOR DE ESTADO SÓLIDO

DATTWYLER RODRÍGUEZ, FERNANDO



UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

DISEÑO DE UNA PLACA DE CONTROL PARA EL MÓDULO DE UN TRANSFORMADOR DE ESTADO SÓLIDO

FERNANDO DATTWYLER RODRÍGUEZ

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELECTRÓNICO

PROFESOR GUÍA	:	DR. MARCELO PÉREZ L.
PROFESOR CORREFERENTE	:	DR. CHRISTIAN ROJAS M.

DICIEMBRE - 2020

A mi familia ...

AGRADECIMIENTOS

Este trabajo marca el fin de mi paso por la Universidad, en el cual muchas personas y momentos marcaron mi vida. Es por esto que en este pequeña nota espero expresar mis más sincera gratitud a cada uno de ellos.

Primeramente quisiera agradecer a mi familia, en especial a mis padres Fernando y Elena quienes me brindaron un apoyo incondicional incluso en los momentos más difíciles, no hay duda que sin ustedes no hubiese podido estar en donde me encuentro hoy. También fueron ustedes quienes me ayudaron a cumplir mi deseo de ir a Suecia a continuar mis estudios, en donde pude ver otra realidad que cambió mis sueños y metas.

A mis amigos del colegio Carlos, Eduardo y Joaquín, con los que formamos un lazo que perdura en el tiempo y siempre que vuelvo a La Serena nos reunimos para compartir nuestras vivencias. A Matías y Elías compañeros de múltiples laboratorios y ramos, estoy seguro que cada uno de esos momentos, incluyendo los malos ahora forman parte de buenas anécdotas. A mis compañeros del Powerlab, con los que siempre encontrábamos un buen momento para poder distendernos y pasar un buen rato. A Diego y Javier a quienes conocí más ya en la etapa final y que con las juntas que teníamos también me llevaron a ver las cosas de forma distinta. A Leonardo con quien compartimos en las Academias tecnológicas, Difusión, Powerlab y terminamos desarrollando nuestras Memorias de forma complementaria, además que su familia me acogió como si fuera uno más de ellos; hay mucho más pero quisiera reservármelo.

A los profesores de física René y Oscar quienes me dieron la confianza para ser su ayudante y que sin duda es parte importante de mi formación. Al profesor César que a pesar de su carácter, me dió una visión diferente sobre la pedagogía y también agradezco su preocupación para que el proceso de intercambio fuera exitoso.

Finalmente se encuentran los profesores Marcelo y Christian quienes me supervisaron durante esta Memoria y me ayudaron para que esta pudiera salir adelante a pesar de las dificultades presentes debido a la pandemia. Que este trabajo pudiera tener esta calidad, se las debo a su dedicación y tiempo a la hora de realizar sus correcciones y de darme consejos en su elaboración.

Fernando Dättwyler Rodríguez

RESUMEN

La generación, transmisión y distribución de energía son las tres partes de un sistema eléctrico de potencia, en las cuales el transformador cumple un rol fundamental permitiendo transmitir la electricidad en grandes distancias con un alto nivel de eficiencia.
Recientemente, los avances de la electrónica de potencia y la implementación de redes inteligentes (Smart Grids), ha traído como consecuencia la popularización del uso de los transformadores de estado sólido (SST). La función de los SSTs es alcanzar el voltaje de la red usando un transformador de alta frecuencia con el fin de reducir el volumen y el peso de un transformador tradicional. Por otro lado, los SSTs permiten la regulación de voltaje y corriente, pudiendo así controlar el flujo de potencia bidireccionalmente y compensar fallas.
En este proyecto se diseñará una tarjeta para el control de un módulo SST, el cual considera la captura y procesamiento de datos provenientes desde el hardware de potencia, la comunicación con el sistema de control externo y la generación de disparos para los semiconductores. Asimismo, es diseñado el sistema lógico que permite unir los elementos mencionados. El diseño PCB de la tarjeta es realizada en ALTIUM DESIGNER y el sistema lógico es implementado en la FPGA LATTICE MACHXO2-HC2000. En este documento se encuentran los esquemáticos y lista de componentes necesarios para el montaje de la tarjeta, además de los códigos del sistema lógico implementado.

Palabras Clave. Transformador de estado sólido, Diseño PCB, Dispositivos lógicos programables, VHDL.

ABSTRACT

Generation, transmission and distribution are the three main parts of an electrical power system, in which the transformer has a fundamental role, allowing transmission over long distances with high efficiency.
Recently, due to the development of power electronics and the implementation of Smart Grids, the use of solid-state transformers (SST) has been popularized. The function of the SSTs is to reach the grid voltage using a high-frequency transformer in order to reduce the volume and weight of a traditional transformer. Moreover, SSTs allow the current and voltage regulation, making bidirectional power flow and fault compensation possible.
In this project, a control board for a SST module is designed, which considers the capture and processing of data coming from the converter, the communication with the external control system and generation of the switching pulses. Likewise, the logical system that links the mentioned elements is designed. The PCB layout is designed in ALTIUM DESIGNER and the logic system is implemented in the FPGA LATTICE MACHXO2-HC2000. This document contains the schematics and bill of materials necessary for the assembly, besides the codes for the logic system implemented.

Keywords. Solid-State Transformer, PCB desing, Programmable logic devices, VHDL.

Índice de Contenidos

Índice de Tablas

Índice de Figuras

1 Introducción

La Memoria de Titulación "Diseño de una tarjeta de Control para el módulo de un Transformador de Estado Sólido", busca realizar mediante el uso de software, el diseño de una tarjeta electrónica que permita controlar el flujo de potencia de un transformador de estado sólido (SST).
Esta tarjeta de control recibe las mediciones de voltaje, corriente y temperatura para el procesamiento y posterior cálculo de los pulsos de disparo que actúan sobre los semiconductores. El diseño además incluye una conexión por fibra óptica para la adición de un elemento de control externo del tipo 'Hardware in the loop' (HIL) que permita simular situaciones o entregar referencias al sistema.
El proyecto comienza con una revisión de las topologías de los transformadores de estado sólido y sus aplicaciones, con esto se obtendrán los requisitos necesarios para su control. A partir de esto, se realizará un estudio de las principales familias lógicas programables como lo son microcontroladores, FPGAs y DSPs. Con la comparación de las familias existentes en el mercado, se evaluará él o los dispositivos que satisfacen los requerimientos de control y comunicación. Respecto a este último punto se hará un resumen de los principales protocolos de comunicación serial con sus usos y comparaciones.
Tomando en consideración la información recopilada acerca de los requerimientos de la tarjeta, presupuesto y desempeño del circuito, se pasará a la etapa de compra de componentes. En paralelo, se realizará el diseño en software y la programación de los integrados en simuladores.
El proyecto culmina con la finalización del diseño de la tarjeta, que incluye el sistema digital y el hardware que permite la posterior implementación del control.

1.1. Contribuciones

La solución aplicada en el diseño del hardware de control para este convertidor modular, permitirá a este regular el voltaje, corriente y dirección del flujo de potencia, las cuales son funciones necesarias para su operación. Además se encuentra el aprendizaje involucrado en el diseño en software de tarjetas electrónicas y el sistema digital que permite comunicar, integrar y realizar las acciones antes mencionadas.

1.2. Objetivo General, Específicos, Alcances y Limitaciones

El objetivo de la memoria es diseñar una tarjeta para el control de un módulo SST y la programación asociada a la captura y procesamiento de los datos de medición, configuración de la línea de comunicación y cálculo de los disparos para el hardware de potencia.

1.2.1. Objetivos específicos

Los objetivos específicos se establecen por cada capítulo y son listado a continuación:

- Recopilar información de topologías y estructura del control del transformador de estado sólido.
- Diseñar tarjeta de pruebas y simular de los integrados de la tarjeta.
- Definir módulos, sub-módulos y flujo de señales de la tarjeta de control.
- Realizar pruebas preliminares de comunicación y adquisición de datos para los integrados de la Tarjeta de pruebas.
- Elección de componentes, realización de esquemáticos y diseño en PCB de la tarjeta de control.
- Medir y obtener resultados del diseño implementado.

1.2.2. Alcances

Esta Memoria sólo contempla el diseño y construcción de la tarjeta de control. El hardware de potencia para la cual se dedica, ya se encuentra diseñada por otro equipo del proyecto.

1.2.3. Limitaciones

Las tareas a desarrollar en esta memoria se encuentran insertas en un proyecto FONDECYT (N°1181839), el cual financia y tiene por objetivo estimular el desarrollo de investigación científica y tecnológica del país. El profesor a cargo tiene la tarea de determinar y organizar los fondos adjudicados para este.
La Memoria contempla una duración de un año en donde se deben cumplir todos los objetivos listados anteriormente. Esta se desarrolla durante los años 2019-2020 los cuales se han visto marcados por la pandemia del COVID-19, la cual ha provocado medidas de confinamiento, entre las cuales se encuentra el cierre de la Universidad y sus laboratorios. Por lo que tareas experimentales y el montaje de la tarjeta se han visto acotadas y reemplazadas por otras de simulación. Esto para poder cumplir los plazos y condiciones de la Titulación.

2 Estado del Arte

En este capítulo se recopilará la información necesaria para sentar la bases técnicas que involucran el tema de esta Memoria. Este se encuentra dividido en tres secciones, la primera trata sobre los Transformadores de estado sólido en donde se exponen sus topologías y necesidades de control. La segunda parte trata sobre los dispositivos lógicos programables que son comúnmente utilizados para este tipo de aplicaciones y sus comparaciones. Finalmente se revisan los buses de comunicación serial, para los cuales se exponen sus principales características.

2.1. Transformadores de estado sólido

El desarrollo que han tenido los semiconductores de potencia en las últimas décadas han incrementado la presencia de estos en aplicaciones de alta potencia, industriales y transmisión de energía. A pesar de esto, la tecnología es reciente y nuevas contribuciones por parte de la academia y la industria han sido generadas en la actualidad [1]. El objetivo de esta sección es hacer una revisión de las topologías y estructuras de este tipo de convertidores.

Los transformadores convencionales han sido una pieza clave en los sistemas de transmisión de corriente alterna por su robustez, confiabilidad y capacidad de operar a distintos niveles de voltajes y potencia, además de ser resilientes a fallas. A pesar de esto, nuevas funcionalidades requeridas por la integración de sistemas de energía distribuidos (basados en energía renovable), electromovilidad y sistemas de tracción, suponen desafíos para elementos tradicionales en la conversión de energía [2]. Es por esto que desventajas como el volumen, tamaño, comportamiento pasivo ante armónicos, falta de regulación de voltaje y factor de potencia hacen necesaria una alternativa a los transformadores convencionales. Los SSTs en su estado actual son menos eficientes operando en condiciones nominales y resultan más complejos en su operación. Estos por su parte permiten el flujo bidireccional de potencia, conexión y desconexión de cargas y fuentes de alimentación, además de el control en la calidad de energía [3].
Los convertidores puente-H en cascada fueron el primer convertidor multinivel que fue teorizado a finales de la década de 1960'. La patente del convertidor de cuatro cuadrantes a base de tiristores de W. McMurray [4] que utiliza este tipo de topología, representa el antepasado directo de los SST modernos. Este tipo de convertidores no tuvieron una aplicación industrial hasta llegada la década de 1990', cuando los semiconductores permitieron trabajar a alto voltaje y frecuencia, haciendo posible que la densidad de potencia aumentara y fuera viable la aplicación de estos [5].
Con el pasar de los años los convertidores multinivel se han impuesto como la solución preferida en media tensión [1]. La configuración en serie y salida en paralelo ha servido como base para los primeros convertidores modulares SST de media a baja tensión, dada la capacidad de operación de alto voltaje en la entrada y de corriente en la salida [6]. Empresas como ABB [7], Alstom [8] y Siemens [9, 10, 11, 12] han presentado prototipos para este tipo de modelos.

La literatura ha propuesto diferentes tipos de arquitecturas las cuales tienen una composición de etapas que pueden descritas como [6, 8]:

- **Etapa de entrada de media tensión:** Esta etapa corresponde a un convertidor AC-AC ó AC/DC, el cual interactúa con la red de media tensión de la red general. El convertidor puede participar en la regulación del factor de potencia absorbiendo o inyectando potencia reactiva, reduciendo así las pérdidas de la red asociada al flujo de potencia reactiva y reduciendo las caídas o elevaciones de la tensión.
- **Etapa de aislación:** El convertidor DC/DC de esta etapa reduce el voltaje de entrada y provee aislación galvánica por medio de un transformador de alta frecuencia. Esta aislación evita que la distorsión harmónica y perturbaciones del voltaje provenientes de la red pase al lado de baja tensión y viceversa, por lo que permite la integración de sistemas de generación distribuidos o estaciones de carga de vehículos eléctricos.
- **Etapa de bajo voltaje:** Este convertidor que puede ser AC/AC ó DC/AC, interactúa con el lado de bajo voltaje de la red que normalmente corresponden a las cargas (de una o más fases). El convertidor compensa el desbalance, caídas y contenido armónico que generan las cargas. En esta etapa pueden ser agregadas funciones de mediciones para el monitoreo del consumo, identificación de la carga y control de la generación o consumo del lado de bajo voltaje.

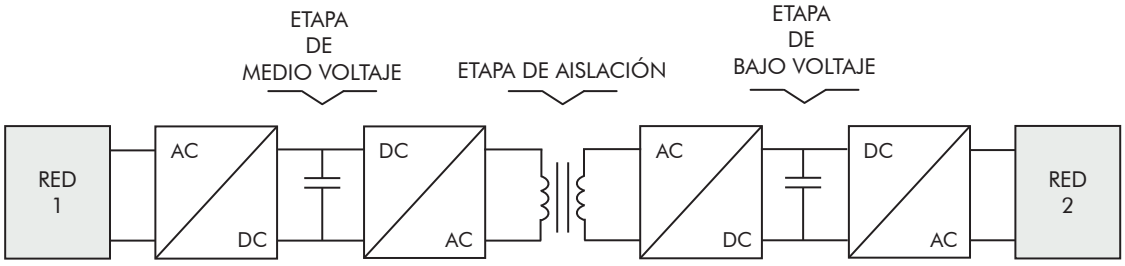


Figura 2.1: Diagrama de etapas SST

Las arquitecturas que han sido desarrolladas pueden ser clasificadas por por su topología, estas son mencionadas a continuación [6, 8]:

- **Arquitectura de aislación simple:** Esta clasificación incluye a todos los SSTs que contienen sólo un transformador monofásico de alta frecuencia, que conecta la etapa de media y baja tensión.
- **Arquitectura de aislación modular:** Esta arquitectura se caracteriza por tener múltiples convertidores DC/DC conectados a través de un transformador monofásico. Esta arquitectura mejora la eficiencia pudiendo apagar celdas de los convertidores en paralelo cuando la carga disminuye. También aumenta la confiabilidad repartiendo la potencia que entrega cada celda. Estos modelos proveen la escalabilidad del voltaje y corriente permitiendo tener flexibilidad en el diseño y mantención.
- **Arquitectura de aislación multipuerto:** Esta clasificación incluye a todos las topologías que incluyen transformadores de devanados múltiples en la etapa de aislación. Dado el acoplamiento magnético, el SST puede intercambiar energía entre los puertos, haciendo un balance de la energía entregada en la carga. Esta capacidad permite redirigir la potencia en caso de fallas y por lo tanto se aumenta la confiabilidad y se reduce el estrés del convertidor, aumentando su tiempo de vida. Esta arquitectura presenta ventajas en términos económicos ya que la utilización de un transformador de múltiples devanados resulta más barata que la compra de múltiples transformadores.
  Esta arquitectura sin embargo no resulta totalmente modular y requiere complejos diseños en el transformador de alta frecuencia en el que debe ser definida de forma precisa la inductancia de dispersión para cada devanado.

En la Figura 2.2 [6] se muestran las familias de arquitecturas explicadas anteriormente.

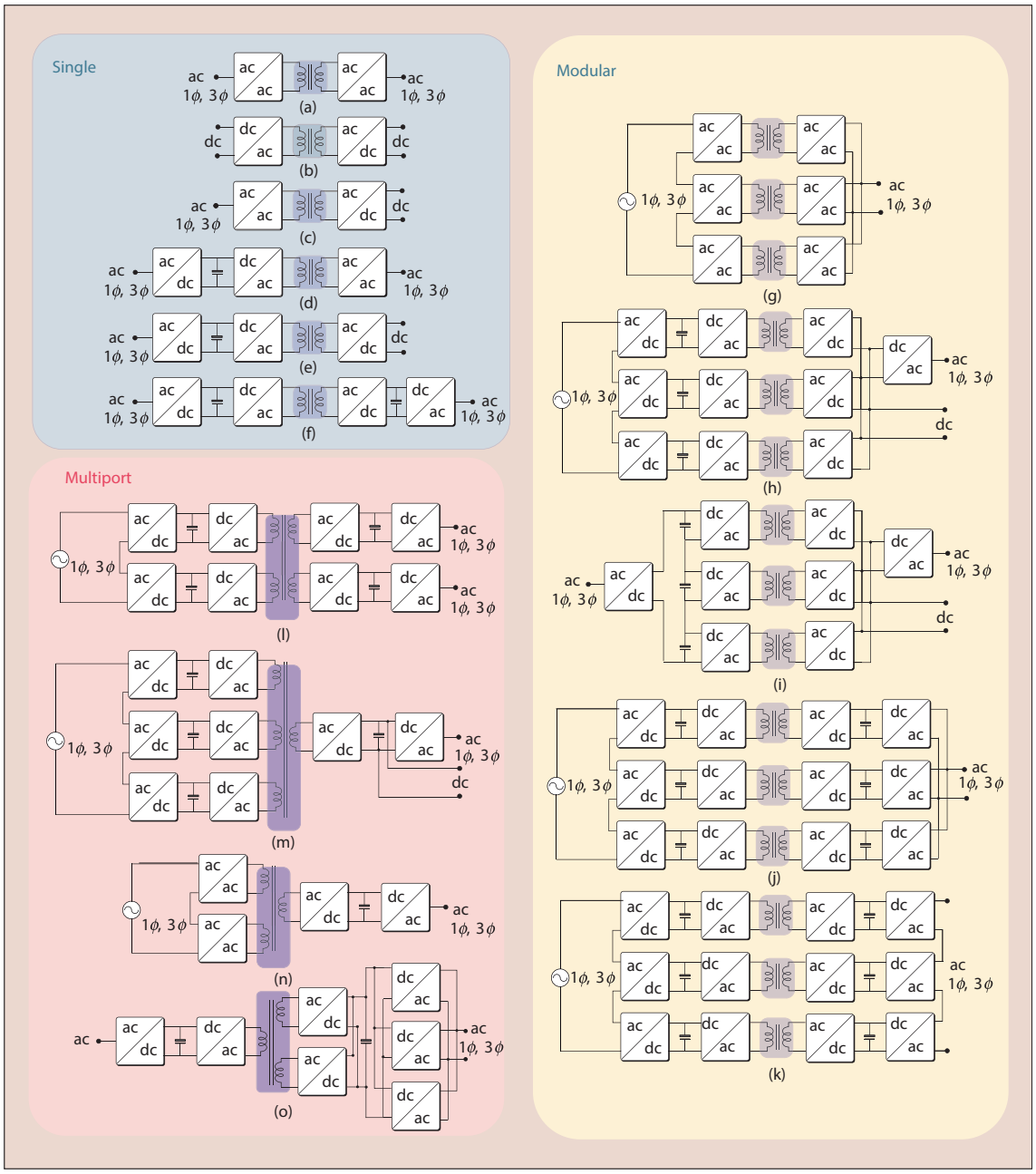


Figura 2.2: Tipos de arquitecturas SST

El convertidor modular presenta grandes desafíos debido a la característica de su control distribuido, lo cual aumenta la complejidad y recae en que el sistema sea sincronizado a través de la comunicación. Cada etapa del convertidor es compuesta por módulos conectados en serie o paralelo como se mencionó anteriormente, donde cada uno posee una unidad de control local.
Es por esto que es necesario organizar la comunicación de forma jerárquica, orientada a maximizar la eficiencia, facilidad en el control, desempeño y minimizar los costos del convertidor. El control jerárquico es organizado en niveles, a los cuales se les brinda un objetivo específico [6].

- El **primer nivel** es responsable del control del voltaje y corriente haciéndose cargo del DC-link y la estrategia de modulación.
- El **segundo nivel** es responsable de la supervisión del control calculando la referencia de las señales de la etapa anterior, logrando así los objetivos del control de flujo de potencia.
- El **tercer nivel** es responsable de la coordinación de todas las etapas del convertidor. Aquí se establece el modo de operación, la adquisición de datos y la estabilidad del convertidor. Además se realiza la relación e interacción con otros elementos de la red, obteniendo datos de sus necesidades.

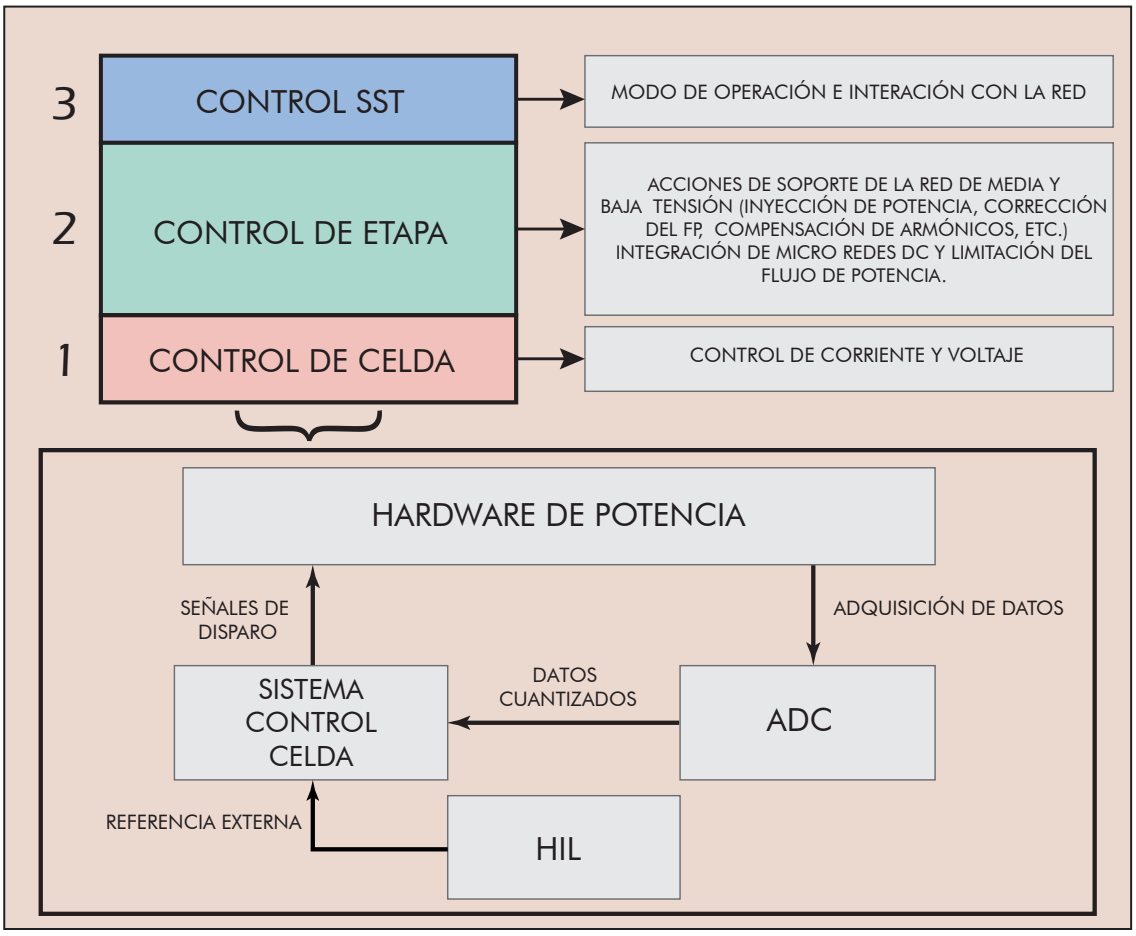


Figura 2.3: Diagrama y estructura de control para el SST

2.2. Dispositivos lógicos programables

Los dispositivos lógicos programables representan el corazón en donde el sistema de control es implementado, hoy día existen numerosas alternativas de solución lo cual puede involucrar combinaciones de estos para llegar a una solución óptima. El objetivo de este apartado es hacer una revisión de la evolución que han tenido las principales familias en la construcción de sistemas lógicos para el control y automatización con aplicaciones industriales.

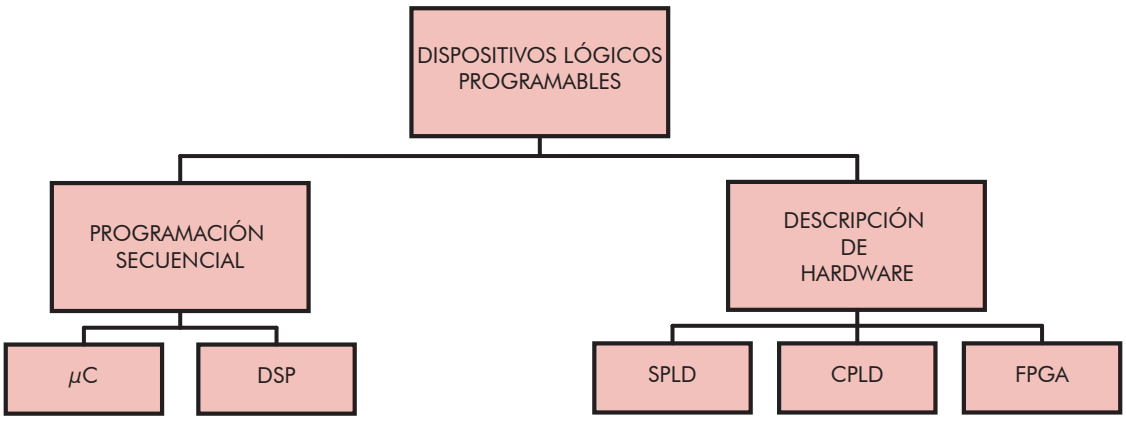


Figura 2.4: Diagrama de los dispositivos lógicos programables

2.2.1. Microcontroladores

Los microcontroladores son sistemas embebidos orientados para el control y automatización de máquinas y procesos. Estos tienen una unidad de procesamiento central (CPU), memoria, puertos de entrada salida (I/O), temporizadores, contadores, conversores análogo-digital (ADC), conversores digital-análogo (DAC), puertos de comunicación, lógicas de interrupción entre otros bloques funcionales. Todos estos bloques se encuentran integrados en un solo chip de bajo consumo y de fácil integración a diseños con propósitos generales y específicos tal como se muestra en la Figura 2.5.

Durante 1970 y 1971 Intel trabajó en el desarrollo de los primeros microprocesadores. Intel 4004 fue el primer procesador de 4-bits, en el cual las instrucciones eran de 8 bits pero estas eran divididas para poder ser procesadas. En 1972 el modelo Intel 4040 aumentaba la capacidad de procesamiento añadiendo 14 nuevas instrucciones con una memoria de 8 kbits y la capacidad de tener interrupciones [13].
En 1974 Texas Instruments introdujo el primer microcontrolador TMS1000. Este modelo incluía una memoria RAM, ROM y puertos I/O en el mismo chip. Durante esos años Intel desarrollo el modelo 8085 el cual podía operar con una fuente de +5V y a una frecuencia de 3MHz.
En paralelo Zilog con su modelo Z-80 mejoró el modelo de Intel utilizando tecnología CMOS. En 1975 Motorola introdujo al mercado su modelo 6800 seguido del 6502 y 6809.
Intel siguió el desarrollo de microcontroladores con la familia MCS-48 la cual fue rápidamente reemplazada por la MCS-51 en 1980, el cual tenía integrada una memoria de 128 bytes. Los controladores con interfaz periférica (PIC) se desarrollaron en la Universidad de Harvard durante 1975 pero no fue hasta 1985 que Microchip los lanzara al mercado, estos utilizaban la arquitectura Harvard y un set de instrucciones reducidas. Ya a fines de la década de 1970' Intel y Zilog introdujeron al mercado microprocesadores de 16 bits. En 1997 ATMEL desarrolló microcontroladores de 8-bits con arquitectura AVR caracterizados por su facilidad y simpleza de programación [13, 14].

CISC y RISC son terminologías típicas cuando se habla de microcontroladores o microprocesadores. El primer término es la abreviación de 'Computadores con Set de Instrucciones Complejas' estos microcontroladores están diseñados para reconocer una gran lista de instrucciones facilitando la flexibilidad y la reducción de líneas de escritura de los programas en lenguaje de ensamblador.
El segundo término es la abreviación de 'Computadores con Set de Instrucciones Reducida', en este caso el microcontrolador sólo reconoce y ejecuta operaciones básicas como sumar o el movimiento de información entre registros. Las operaciones más complicadas se realizan al hacer una combinación de éstas (por ejemplo, la multiplicación se lleva a cabo al realizar adición sucesiva.). La ventaja de esto es que el diseño del microcontrolador necesita menos hardware, reduciendo el coste de producción y la escritura en compiladores optimizados, aumentando la velocidad de ejecución de las instrucciones. Este tipo de microcontroladores son populares dentro de las familias de PIC's de Microchip [14].
El mercado de los microcontroladores se sigue desarrollando mejorando la velocidad y capacidad de sus procesadores y presentando opciones tan diversas como las necesidades de la aplicación. A continuación se listan una serie de compañías las cuales son los principales fabricantes de microcontroladores:

- Microchip
- Texas Instruments
- ST
- NXP
- Renesas
- Maxim Integrated

En las Tablas 2.1, 2.2 y 2.3 son expuestas las principales familias de las marcas Texas Instruments, Microchip y NXP correspondientemente con una breve descripción de sus características principales [15, 16, 17]. En la Figura 2.6 se agrupan las familias de las diferentes marcas y son clasificadas según sus bits de arquitectura.

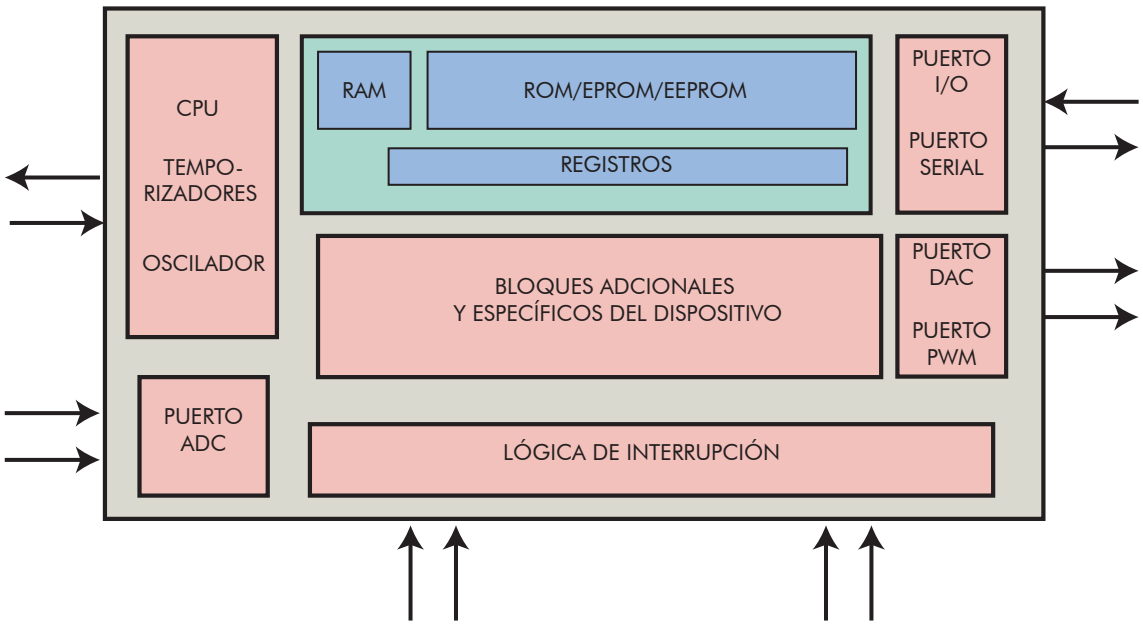


Figura 2.5: Diagrama de los bloques funcionales de un microcontrolador

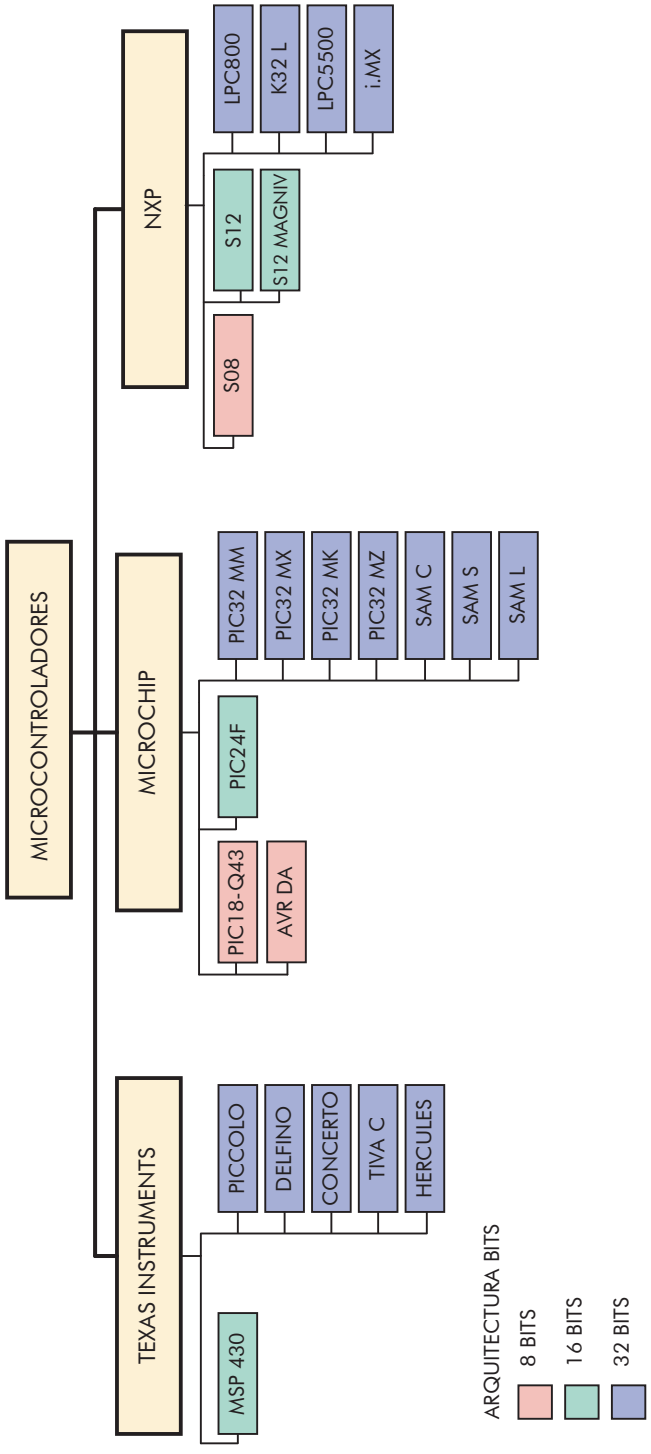


Figura 2.6: Esquema de familias de microcontroladores

Tabla 2.1: Familias de microcontroladores Texas Instruments

Familia	n° Bits	Subfamilia	Características
Piccolo	32	F2802x/F2803x/ F2805x/F2806x	Familia de bajo consumo enfocada a aplicaciones de control en tiempo real desde electrodomésticos a accionamientos industriales. Los dispositivos tienen un procesador de hasta 90 MHz que puede operar aritmética de punto flotante. Incluye un módulo ADC de 12 bits que tiene una capacidad de muestreo de hasta 4.6 MSPS. Además de una gran conectividad mediante puertos UART, SPI, I2C, USB y CAN.
Delfino	32	F2833x/C2834x	Familia de alto desempeño para aplicaciones de electrónica de potencia como inversores, fuentes de poder, radar y sensado inteligente. Los dispositivos tienen un procesador de hasta 300 MHz que opera aritmética de punto flotante. Incluye un módulo ADC de 12 bits con una capacidad de muestreo de 12.5 MSPS. Incluye puertos de comunicación UART, SPI, I2C, USB y CAN.
Concerto	32	F28M35Ex/ F28M35Mx/ F28M35Hx	Familia de microcontroladores de doble núcleo. Contiene un procesador C28x de hasta 150 MHz para un sub módulo de control que comparte las cualidades de las famlias Piccolo. Contiene un procesador Cortex M3 de hasta 100MHz, encargado del sub módulo de comunicaciones. Ambos módulos comparten la alimentación, oscilador y memoria.
TIVA C Series	32	LM4xx5Mx/ F28M35Hx	Familia enfocada a aplicaciones de seguridad, domótica, HMI y control industrial. Poseen un procesador CortexM4F de hasta 80 MHz con posibilidad de modo de bajo consumo. Integra una gran cantidad de módulos para la simplificación y reducción del tamaño de la solución, en los que incluye puertos de comunicación, LDO, PWM y memorias ROM y, RAM y Flash. Soportado por su propio software TivaWare para facilitar la escritura de su código.
Hercules	32	RM4x/ TMS570LS/ TMS470M	Familia de alto rendimiento dedicada a aplicaciones de seguridad industrial y médicas. Con un procesador Cortex R4F de hasta 220 MHz que soporta operaciones de punto flotante. Integra con sus periféricos opciones avanzadas de conectividad, control en tiempo real. La sub familia TMS570LS posee nucleo doble para cumplir las necesidades de seguridad en aplicaciones de trenes, automóviles y aereoespacial.
MSP430	16	MSP430FRxx	Familia de bajo consumo y propósito general orientada a la adquisición de datos y mediciones con módulos ADC y DAC integrados. Modulos de comunicacion SPI, UART, I2C para integración con otros dipositivos. La sub familia CapTIve añade un módulo especializado para botones o sliders capacitivos enfocados a HMI industriales. La sub familia Sistem-on-chip presenta un módulo DSP para el procesamiento de mediciones de flujómetros de ultra sonido.

Tabla 2.2: Familias de microcontroladores Microchip

Familia	n° Bits	Subfamilia	Características
PIC32MM	32	PIC32MMxx	Familia de bajo costo y consumo. Estos dispositivos presentan el puente entre los PIC24 XLP y los PIC32MX. Cuentan con un procesador de 25 MHz el cual puede combinar instrucciones de 16 y 32 bits reduciendo el espacio de la memoria.
PIC32MX	32	PIC32MXxx	Familia de propósito general y bajo costo integra módulos para aplicaciones de audio, gráficas y de conectividad. Incluye una sub familia de bajo consumo con un procesador más potente que la familia MM. Cuentan con procesador MIPS32 MK de hasta 120 MHz y un ADC de 10 bits con una capacidad de muestreo de 1 Msps para aplicaciones industriales. Y hasta cuatro interfaces SPI e I2S para procesamiento de audio.
PIC32MK	32	PIC32MKxx	Familia orientada al control de motores con un gran rango de periféricos de comunicaciones. Esta familia cuenta con un procesador microAptiv de hasta 120 MHz, 7 ADC independientes de 12 bit con capacidad de 3.75 MSPS y 3 DAC de 12 bits. Además contiene un módulo DSP que permite llevar a cabo operaciones de punto flotante.
PIC32MZ	32	PIC32MZxx	Familia de alto rendimiento. Posee un procesador de hasta 252MHz y una memoria flash de hasta 2Mb. Contiene una gran cantidad de periféricos para las comunicaciones USB,CAN,SPI,I2C. Un bloque DSP de doble precisión para operaciones de punto flotante y un hardware de aceleración para aplicaciones de criptografía.
SAMC	32	SAMC20/ SAMC21	Familia que cuenta con un procesador Cortex M0+ de hasta 48 MHz y que puede operar con un voltaje de 2.7-5 V. Poseen una gran cantidad de periféricos de comunicación cubriendo todas las aplicaciones industriales. Además de múltiples modos de operación para el reloj y de bajo consumo.
SAML	32	SAML10/ SAML11/ SAML21/ SAML22	Familia de bajo consumo y propósito general orientada a la adquisición de datos y mediciones con módulos ADC y DAC integrados. Módulos de comunicación SPI, UART, I2C para integración con otros dispositivos. La sub familia CapTIve añade un módulo especializado para botones o sliders capacitivos enfocados a HMI industriales. La sub familia Sistem-on-chip presenta un módulo DSP para el procesamiento de mediciones de flujómetros de ultra sonido.
SAMS	32	SAM70	Familia de propósito general con un procesador Cortex M7 de hasta 300 MHz de alto desempeño. Soporta operaciones de punto flotante. Contiene una gran cantidad de periféricos para cumplir con las tareas de conectividad, control, seguridad e interfaces de usuarios.
PIC24F	16	PIC24xx	Familia de bajo costo y consumo que cuenta con un procesador que llega a 16MIPS. Son escalables y cuentan con un gran número de empaquetados de distintos tamaños. Cuentan con la tecnología eXtreme Low power que permite diferentes modos de bajo consumo.
PIC18-Q43	8	PICFxxQ43	Familia de bajo costo que cuenta con un procesador con un velocidad de 16MIPS, de empaquetado pequeño. Cuenta con módulos ADC y PWM para desarrollar tareas de control en tiempo real. Además de periféricos de comunicación para la conectividad con otros elementos.
AVR DA	8	AVRxxDA	Familia orientada al control en tiempo real y para interfaces táctiles capacitivas, Cuenta con periféricos independientes al núcleo que pueden funcionar en modo de bajo consumo. Con una operación de 5 V para aumentar la inmunidad al ruido.

Tabla 2.3: Familias de microcontroladores NXP

Familia	n° Bits	Subfamilia	Características
LPC800	32	LPC80x/ LPC81x/ LPC82x/ LPC83x/ LPC84x	Familia de bajo costo y consumo con empaquetados pequeños desde 24 pines. Poseen un procesador Cortex M0+ de hasta 30 MHz. Abarcan gran cantidad de aplicaciones de propósito general con módulos ADC/DAC para el control y UART/SPI/I2C para la comunicación con otros dispositivos. La presencia de estos módulos dependen del modelo de la familia. Esta familia presenta la transición de los antiguos modelos de microcontroladores de 8-16 bits a los de 32.
K32 L	32	K32 L3/ K32 L2	Familia de microcontroladores diseñados para aplicación de bajo consumo y seguridad enfocadas en un ambiente de IoT. Contienen un procesador Cortex-M0 o M4 de hasta 72 MHz, además cuentan con detección de manipulación en los pines que habilitan registros de interrupción y aceleradores optimizados para tareas de criptografía.
LPC5500	32	LPC557x/ S7xLPC55S6x/ LPC553x/ S3xLPC552x/ S2xLPC551x/ S1x	Familia de microcontroladores pensada en ser la línea principal de producción para el mercado. Dependiendo de la subfamilia presentan doble núcleo y aceleración para procesamiento de señales con lo que se reduce hasta 10 veces el tiempo de tarea por ciclo de reloj. La familia posee la tecnología de 40 nm NVM que permite ser escalables en memoria y tamaño los empaquetados de los dispositivos.
i.MX	32	i.MX RT1170/ i.MX RT1064/ i.MX RT10xx/ i.MX RT600/ i.MXRT500	Familia de alto desempeño que contiene núcleos múltiples que combina procesadores Cortex M7 y M4, para paralelizar el control de interfaces gráficas y el sistema de control. Contiene una arquitectura que flexibiliza el uso de memoria externas y facilita la conectividad Wireless vía Wifi, Bluetooth, Zigbee y Thread.
S12	16	S12X	Familia diseñada para aplicaciones en la industria automotriz. Los dispositivos de esta familia se caracterizan por tener empaquetados pequeños, por lo que dependiendo de la subfamilia se pueden ver microcontroladores que no presentan todos los módulos de comunicaciones, ADC o capacidad de memoria.
S12 Magniv	16	S12Zx/ S12Vx	Familia diseñada para el control de motores, actuadores y drivers para los convertidores. Tienen un núcleo S12Z de hasta 50 MHz e incluyen interfaz CAN y LIN para la comunicación. Los modelos pueden operar hasta 20 V, integrando reguladores de tensión internos de 5 y 12 V y seis unidades para el control de Gate drivers MOSFETs reduciendo así el número de componentes para el diseño, además que los dispositivos se destacan por el tamaño pequeño de sus empaquetados.
S08	8	SO8xx	Familia de bajo costo en donde se encuentran productos de empaquetado robusto para ambiente industrial, bajo consumo y aplicación automotriz. Dependiendo de la sub familia presentan módulos de comunicación especializados como CAN, LIN o SPI y capacidad en su memoria.

2.2.2. Procesadores digitales de señales

Estos dispositivos comúnmente conocidos por su abreviación DSP, fueron creados en los comienzos de la década de 1980', ayudando al desarrollo de áreas que necesitan un alto nivel de cálculo como lo son la militar, aeroespacial o médica. Áreas en que el que el procesamiento de señales

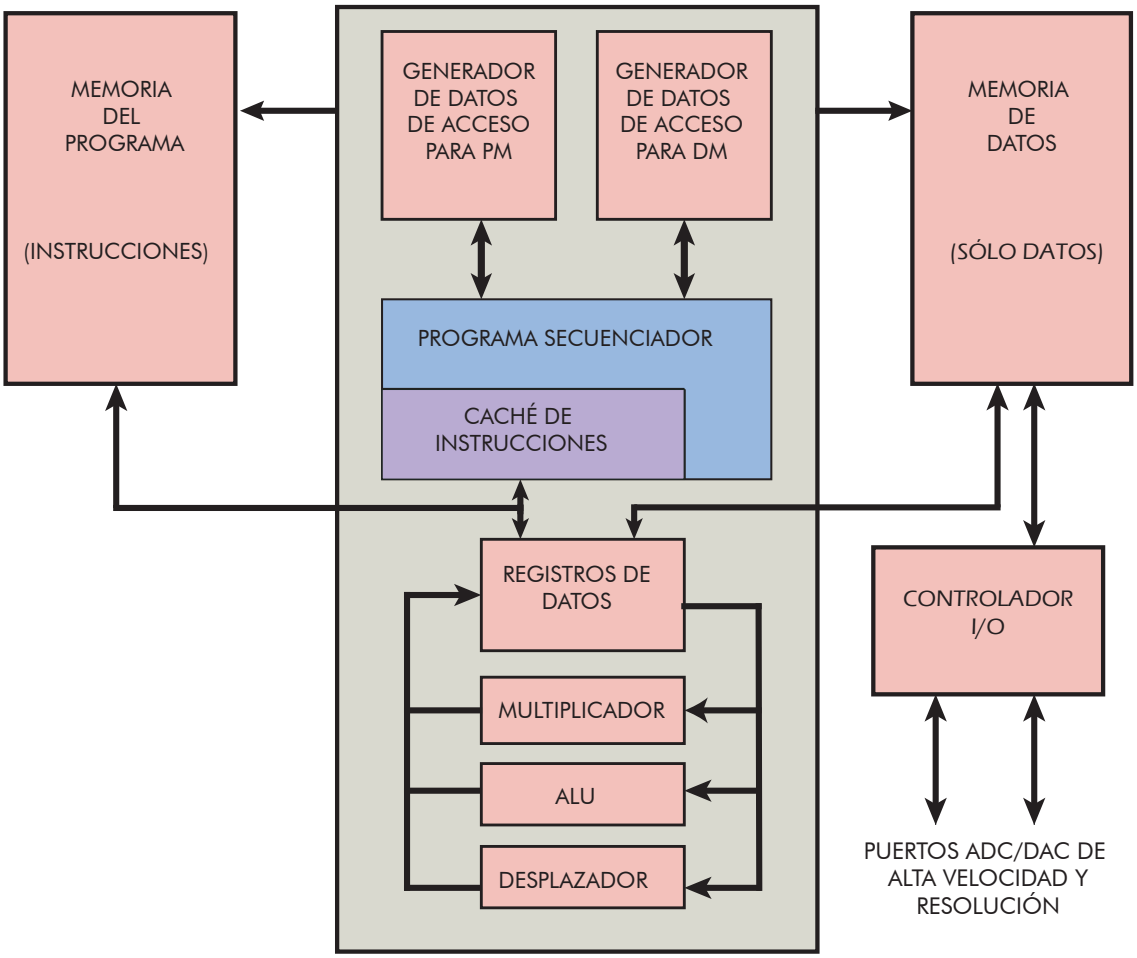


Figura 2.7: Diagrama simplificado de la arquitectura de un DSP

y la baja latencia del cálculo son parte de sus requerimientos.
Los primeros ejemplares fueron introducidos al mercado por Texas Instruments, los cuales realizaban un excelente trabajo al ejecutar operaciones de matemática de punto flotante, estos eran difíciles de programar usando sólo lenguajes de bajo nivel, dada la complejidad de las instrucciones.
El modelo i860 de Intel redujo el set de instrucciones usando una arquitectura RISC, lo que pudo llevar a los DPSs al siguiente nivel. Este modelo podía ser programado usando lenguaje C, facilitando las operaciones matemáticas de punto flotante. Como resultado el modelo i860 fue rápidamente popularizado para aplicaciones de radar e inteligencia de señales.
Aunque Intel dominó el mercado temporalmente, Freescale (actualente NXP) desarrolló una nueva tipo de arquitectura (Power Arquitecture) y el concepto de motor vectorial. Lo cual permitió que sus modelos fueran capaces de desempeñar una gran cantidad de operaciones de punto flotante por ciclo de reloj. Este conjunto de instrucciones fueron llamadas como AltiVec y fueron usadas en los procesadores de IBM, Apple y Motorola.
Luego dado factores económicos, Freescale cambió su enfoque en el comienzo de los 2000' a la telecomunicación y otros mercados siguiendo su desarrollo de AltiVec. A finales de la década Intel ganó una gran presciencia en el mercado en el área de la defensa y seguridad mientras que Freescale tuvo un nuevo auge con la introducción de los chips con multi-núcleo [18]. Hoy el futuro de los DSP's recae en los procesadores multi-núcleos, como lo son los dispositivos de la familia Xeon de Intel los cuales tienen más de 60 procesadores integrados. Esto ofrece núcleos con procesadores gráficos incluidos y una gran eficiencia en términos de operaciones por watt de potencia consumida.
Otra rama de desarrollo es la combinación entre la línea más tradicional de los DSPs con otros

sistemas embebidos, como las FPGA's o microcontroladores. Usando núcleos ARM para tener procesadores gráficos incluidos en el mismo empaquetado. Haciendo estos dispositivos más autónomos, poderosos y flexibles en el diseño de sistemas digitales.
Otro avance es la inclusión de memorias internas en los chips que son usados en tarjeta, llegando a un almacenamiento del orden de los Gigabytes, disminuyendo inmensamente la latencia en sus operaciones.

Actualmente, la frontera entre microcontrolador, microprocesador y DSP es cada vez más diluida. Es fácil encontrar en el mercado microprocesadores y DSPs que incorporan memoria y periféricos internos o microcontroladores con CPUs tan potentes como las de un DSP. A veces la diferencia entre ellos es nula, y el nombre microcontrolador, microprocesador o DSP se convierte más que nada en una cuestión de marketing [19].
Los principales desarrolladores y fabricantes de DSPs son:

- Microchip
- Texas Instruments
- NXP
- Intel
- ON Semiconductor

En la Tabla 2.4 se muestra una recopilación de familias de las marcas Texas Instruments, Microchip y NXP [20, 21, 22]. En la Figura 2.8 se agrupan las familias de DSPs de las marcas estudiadas.

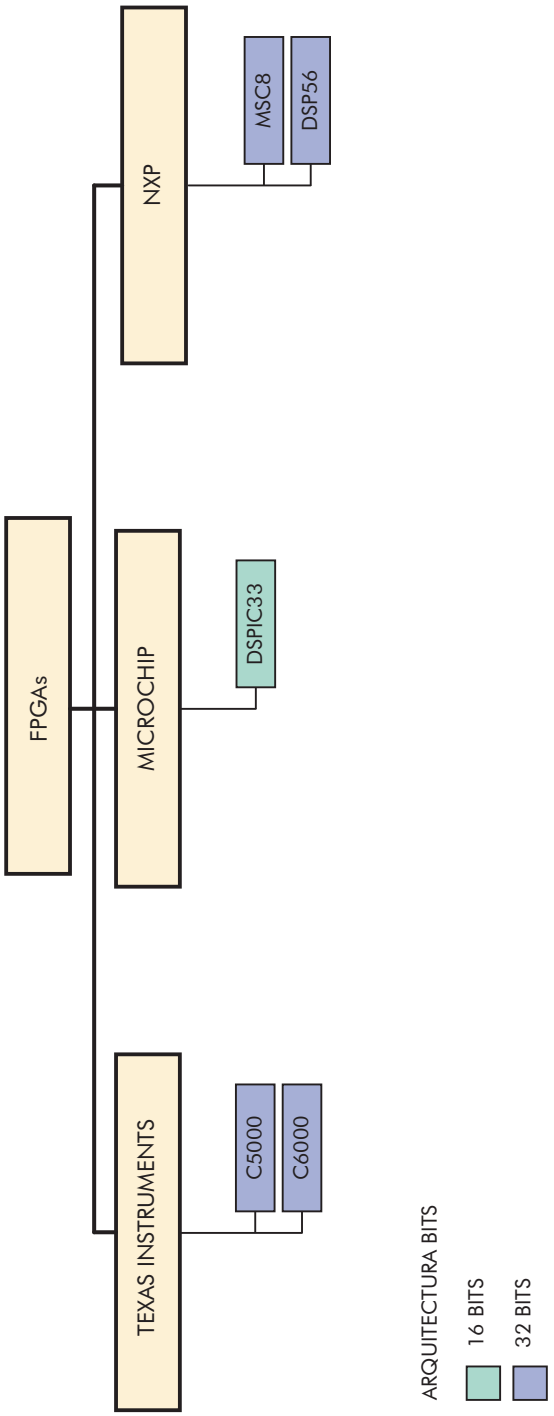


Figura 2.8: Esquema familias DSPs

Tabla 2.4: Familias de DSPs

Marca	Familia n° Bits	Características
Texas Instruments	C5000 32	Corresponde a la familia de DSP de bajo consumo con un procesador de 300 MHz que trabaja con operaciones de punto fijo. Pensada para aplicaciones gráficas, audio y voz, posee hardware de aceleración para FFT y periféricos para comunicación serial.
Texas Instruments	C6000 32	Familia multinúcleo de alto rendimiento con una velocidad de hasta 1GHz. Diseñado para aplicaciones de comunicaciones, control industrial, audio y transporte. El procesador ARM se ocupa de los periféricos de control, comunicación y captura de datos y el procesador C66 de las tareas de cálculo de punto fijo y flotante.
Microchip	dsPIC33 16	Familia de núcleo doble que opera a una velocidad máxima de 200 MHz. Diseñada para el control de motores el cual cuenta con periféricos ADC y PWM para este propósito. Familia de productos escalables con diferentes tamaños y cantidad de pines para sus empaquetados.
NXP	MSC8 32	Familia de seis núcleos, los cuales operan a 1 GHz, para tareas de alto rendimiento. Diseñado para tareas de comunicaciones inalámbricas, médicas, aeroespaciales. Contienen módulos dedicados para el procesamiento de DFT y FFT y subsistemas que soportan protocolos de redes para la transmisión correcta de datos.
NXP	DSP56 24	Familia de DSP de doble núcleo que operan a una velocidad de 250 MHz. Diseñados para tarea de audio y video soportando los decodificadores de Dolby, THX y DTS entre otros. Esta familia cuenta con productos escalables en el número de pines permitiendo tener dispositivos de tamaño reducido.

2.2.3. FPGAs

Una matriz de compuertas lógicas o FPGA es un circuito integrado que puede ser configurado después del proceso de manufactura. La FPGA se configura por medio de un lenguaje de descripción de hardware (HDL), lo cual interconecta los bloques y compuertas lógicas para la construcción de sistemas combinacionales, los cuales procesan las entradas para obtener señales de salida.

La industria de las FPGAs nació en la década de 1980' a través del desarrollo de los dispositivos lógicos programables (PLD) y las memorias programables de sólo lectura (PROM).
Estos dispositivos debían ser cableados durante el proceso de manufactura, por lo que no podían ser reprogramados.
El primer dispositivo lógico reprogramable fue el modelo EP300 de la compañía Altera en 1984. Este tenía una lámina de vidrio, que permitía que la luz ultravioleta llegase a la memoria programable (EPROM), logrando así que la configuración del dispositivo pudiera ser cambiada cuando fuera necesario.
El desarrollo de los dispositivos programables fue extendido por LuVerne R. Peterson y David W. Page quienes a mediados de la década de 1980', crearon las patentes para los bloques lógicos, compuertas y arreglos lógicos programables.
A finales de la década, la FPGA fue creada a través de los experimentos sugeridos por Steve Casselman, donde su propuesta era crear una computadora con más de 600.000 bloques reprogramables. Su trabajo fue exitoso y su creación fue patentada en 1992. A mediados de 1990', el desarrollo de la FPGA tuvo un gran impulso y revolucionó la industria en áreas como las telecomunicaciones, la automatización, la seguridad y el procesamiento de señales. En las cuales la FPGA ofrece soluciones

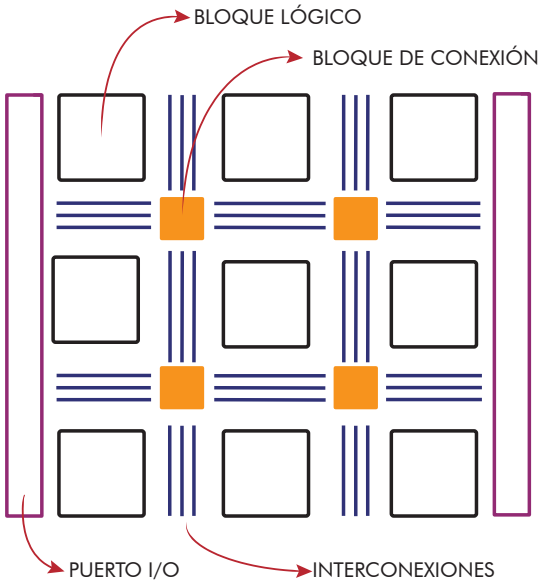


Figura 2.9: Arquitectura de una FPGA

de bajo costo y flexibilidad [23, 24].
La FPGA a diferencia de los microcontroladores y DSPs, procesa las señales de forma paralela siendo mucho más rápida para realizar tareas, ocupando máquinas de estados finitos. Esta no tiene un set de instrucciones preconfigurado por lo que ofrece más opciones para su programación y diseño.
Es por ello que presenta una complejidad relativa para su configuración. El tiempo necesario para la traducción del código de descripción y selección del hardware es mayor que la síntesis en microcontroladores. Por otro lado, estos dispositivos tienden a consumir más energía al no estar optimizados y sus modelos son más voluminosos y caros en comparación a los microcontroladores [25].

El mercado en 2020 se caracteriza por tener productores de FPGA's de propósito general y específicos dentro de los cuales se encuentran:

- Xillinx
- Intel
- Lattice
- Microchip

Las FPGAs son ocupadas en una basta cantidad de aplicaciones, una recopilación de familias existentes en el mercado se exponen en las Tablas 2.5 y 2.6.

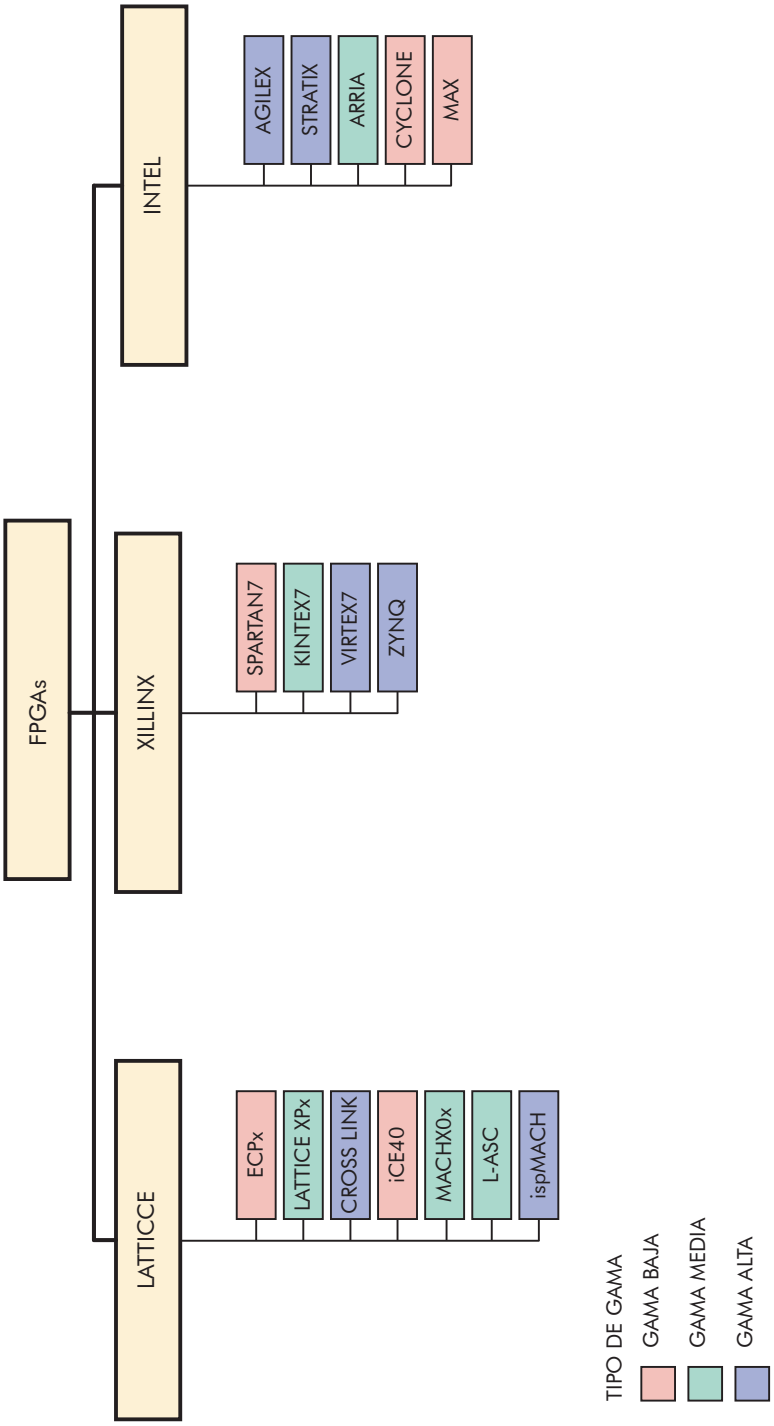


Figura 2.10: Esquema familias FPGAs

Tabla 2.5: Familias FPGA XIllinx e Intel

Marca	Familia	Características
Xillinx	SPARTAN7	Familia dedicada a ofrecer el mejor rendimiento energético con empaquetados pequeños. Usa la tecnología de 28 nm incluyendo un procesador interno y soporte para memorias externas. Incluye ADC para aplicaciones industriales, de propósito general, conectividad y visión embebida.
Xillinx	KINTEX7	Familia de gama media que usa tecnología de 28 nm. Entrega alto rendimiento usando bloques DSP en un empaquetado optimizado. Incluye periféricos para redes conectadas por fibra óptica y redes inalámbricas.
Xillinx	VIRTEX 7	Familia de alto rendimiento incluye bloques DSP, I/O de banda ancha para aplicaciones de redes 10-100G, radar y prototipado ASIC.
Xillinx	ZYNQ	Familia de alto rendimiento que posee un o dos núcleos Cortex A9 para el procesamiento junto con la lógica programable de la FPGA. Incluye memoria interna y módulos para agregar memorias externas junto con periféricos de comunicación de alta velocidad y bloques DSP. Fue diseñada para ofrecer soluciones en tareas de video vigilancia, asistencia automotriz, sistemas inalámbricos y automatización.
Intel	AGILEX	Familia dedicada a ofrecer soluciones a centro de datos, procesamiento autónomo de datos y redes. Cuenta con un procesador DSP que permite realizar operaciones de punto fijo y flotante 32 bits de alta precisión. Integra memoria y la opción de conectar una memoria externa de 16 GB soportando un ancho de banda de 512 GB/s. Posee protección contra escritura y de sus periféricos.
Intel	STRATIX	Familia de alto rendimiento que puede incluir un procesador Cortex A53 y bloque DSP para resolver operaciones de punto flotante. Sus aplicaciones están enfocadas en el área de la Ciber seguridad, acelerador de tareas en Data Centers, línea de comunicaciones, radar y redes.
Intel	ARRIA	Familia de rango medio que utiliza tecnología de 20 nm. Posee un núcleo doble Cortex A9 e incluye bloques DSP. Ofrece una amplia cantidad de periféricos embebidos , transceptores de alta velocidad y controladores de memoria. Sus aplicaciones se encuentran en las telecomunicaciones, centros de datos, médicas y militares.
Intel	CYCLONE	Familia de bajo costo y consumo que usa la tecnología de 28 nm. Integra transceptores, procesador ARM y periféricos dependiendo de la subfamilia. Sus aplicaciones se ven pensadas para el control de motores, expansión de puertos I/O e interfaces.
Intel	MAX	Familia de bajo costo que integra gran cantidad de puertos I/O en un empaquetado reducido. Incluye ADC, bloques DSP y un procesador embebido para el proceso de tareas de control industrial y automotriz. Adaptadores de memoria externa para el almacenamiento de datos, óptimo para para video y otras aplicaciones embebidas.

Tabla 2.6: Familias de FPGAs Lattice

Marca	Familia	Características
Lattice	ECP5 / ECP5-5G	Esta familia provee soluciones de bajo costo y consumo para implementaciones de vídeo e imagen o aplicaciones de alto volumen como sistemas de cámaras industriales, micro servidores o aplicaciones automovilísticas.
Lattice	ECP3/2	Familias de bajo costo y consumo con gran nivel de integración incluyendo bloques DSP de alto rendimiento y hasta 70000 LUTs. Soportan la mayoría de protocolos de comunicación serial de propósitos generales.
Lattice	Crosslink	Familia que soporta una gran cantidad de protocolos e interfaces para sensores y monitores. Los dispositivos están basados en la tecnología de 40nm combinando la flexibilidad y el bajo consumo. Usando módulos preconfigurados facilitando la programación.
Lattice	iCE40	Familia de bajo consumo con empaquetados pequeños ($< 6\ mm^2$). Cuenta con transceptores internos para cámaras u otros dispositivos. Pueden ser programados en una memoria no volátil a través de I2C y SPI.
Lattice	MachXO(0/2/3)	Familias orientadas al control y seguridad, cuentan con empaquetados escalables y resistentes al ambiente industrial. Los modelos cuentan con controladores SPI e I2C preimplementados junto con PPL, memorias Flash bloques RAM distribuidos.
Lattice	ispMACH	Familias de bajo consumo que operan a alta velocidad. Tienen un sistema de enrutamiento especializado para tener tiempos de respuestas precisos.

En la Tabla **??** se resume la comparación entre las familias analizadas en la sección. La información de la tabla es sólo referencial dado que existen dispositivos que pueden incluir capacidades que por lo general no es parte de su familia.

Tabla 2.7: Comparación de dispositivos lógicos programables

	Familia Lógica Programable		
Característica	Microcontrolador	DSP	FPGA
Arquitectura en bits	8/16/32	16/32	-
Consumo de Energía	Bajo	Medio/Alto	Medio/Alto
Capacidad de integración	Alta	Baja	Alta
Programación de interrupciones	Si (Registros Dedicados)	Si (Registros Dedicados	Si (Lógica combinacional)
Paralelismo	No	Si (Pseudo paralelismo a través de múltiples procesadores)	Si
Memoria interna	Si	Sólo algunos modelos	Sólo algunos modelos
Tamaño del empaquetado	Escalables con facilidad de encontrar empaquetados pequeños	Medianos/Grandes	Escalables
Precio	Bajo	Medio/Alto	Medio/Alto

2.3. Comunicación Serie

Para comunicar a diferentes dispositivos es necesario establecer un bus de comunicación. Este regirá sobre la tasa de transmisión y la relación que tendrán para el envío y recepción de datos. A continuación se muestran los principales buses de datos que son implementados por comunicación serie.

2.3.1. SPI (Serial Peripheral Interface)

Bus de comunicación en serie de tipo síncrono, usado ampliamente para la comunicación entre dispositivos y periféricos como sensores, ADCs, DACs y memorias. Este bus consta de dos líneas de datos, una línea de reloj y una línea de selección de esclavo.
Los términos para cada señal del protocolo son presentadas a continuación [26, 27].

- **Maestro:** Dispositivo que proporciona la señal de reloj para la comunicación, controla además el flujo de datos y estado de cada dispositivo.
- **Esclavo:** Dispositivo de comportamiento pasivo y dependiente de los comandos del maestro.
- **MOSI:** Master Out Slave In, es la línea por la cual el maestro envía datos a sus esclavos.
- **MISO:** Master In Slave Out, es la línea por la cual los esclavos envían datos al maestro.
- **SCK:** Línea por la cual el maestro transmite la señal de reloj.
- **SS:** Slave Select, línea por la cual el maestro selecciona el esclavo con quien comunicarse.

En este tipo de comunicación sólo se puede tener un maestro que controle uno o varios esclavos, por lo que todo el flujo de datos, es manejado por este. Un diagrama de la comunicación es presentado en la Figura 2.14.

Además algunas de sus características son:

- Este bus puede llegar a una velocidad de comunicación entre el rango de 10-20 Mb/s.
- Su implementación en hardware es sencilla al no necesitar resistencias pull-up.
- Permite cuatro modos de funcionamiento dependiendo de la selección de fase y polaridad del reloj para la lectura y escritura de datos.
- Debe pasar por una interfaz para poder transmitir el bus por RS-232, RS-485 o CAN.

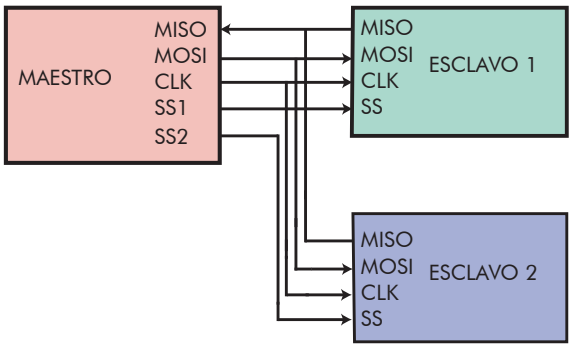


Figura 2.11: Diagrama de comunicación SPI

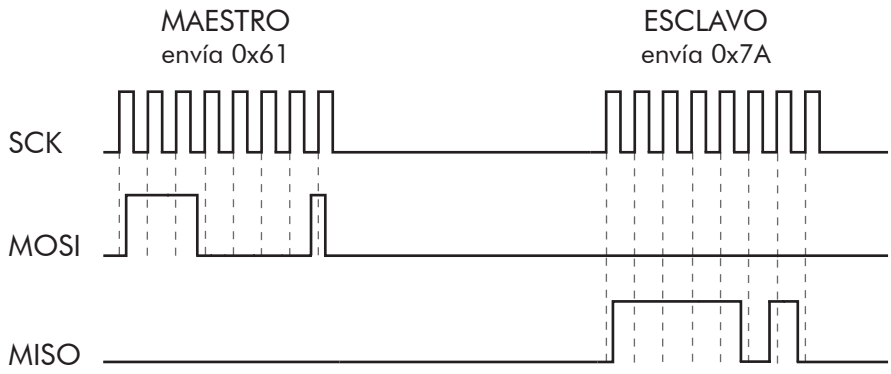


Figura 2.12: Bus de transmisión SPI

2.3.2. I2C

El bus I2C fue introducido en 1982 por Philips para la comunicación entre circuitos integrados. Este ha tenido varias actualizaciones durante los últimos años y es en el año 2012 con la versión V.4 fue posible llegar a velocidades de transferencia unidireccionales de hasta 5 Mb/s.
I2C utiliza dos cables para todo el proceso. El protocolo I2C puede admitir más de un maestro y esclavo.

La terminología asociada al protocolo es:

- **SDA:** Es la línea por la que se mueven los datos entre los dispositivos.
- **SCL:** Es la línea de los pulsos de reloj que sincronizan el sistema.
- **VDD:** Línea de alimentación
- **Rp:** Resistencias 'Pull up' que deben ser agregadas a la línea.

El protocolo funciona de la siguiente manera. Cada dispositivo envía y recibe datos usando solo un cable que es SDA. El cable SCL mantiene la sincronización entre los dispositivos a través del reloj común que proporciona el maestro activo [28]. Cada maestro se encuentra constantemente monitoreando las señales SDA y SCL, determinando si el bus está ocupado o no, En el caso que dos o más maestros llegarán a transmitir al mismo momento, si uno de ellos detecta que SDA está en '0' y el valor a escribir es '1', asume que otro maestro está activo y termina la transferencia de bits, esto se le llama proceso de arbitraje.

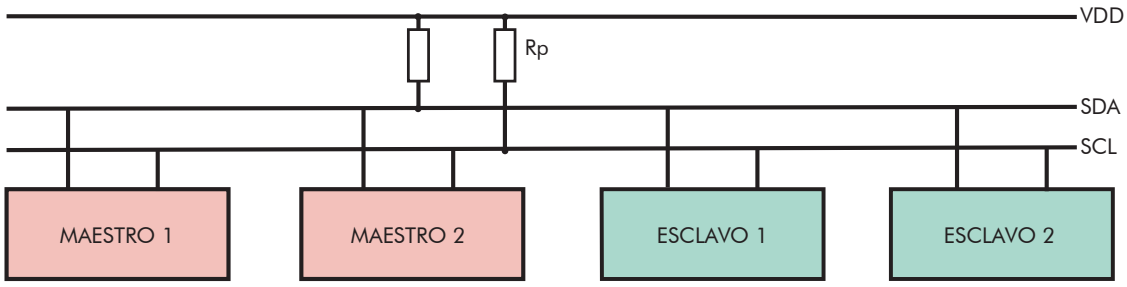


Figura 2.13: Diagrama de comunicación I2C

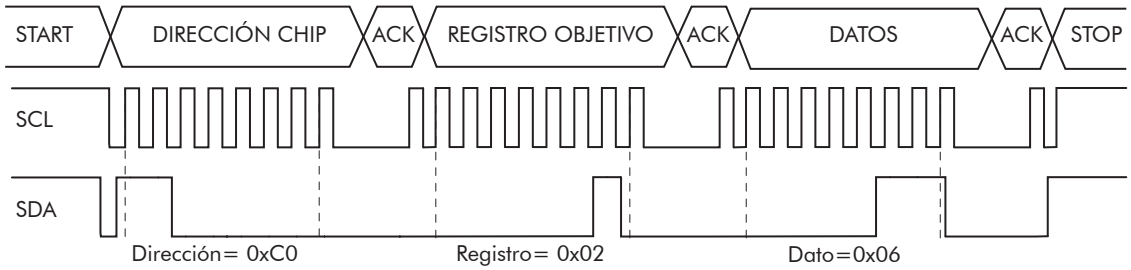


Figura 2.14: Bus de transmisión I2C

2.3.3. Universal Asynchronous Receiver Transmitter (UART)

El bus UART es un estándar de comunicación serial. El registro de datos comienza con un bit de partida que es seguido con 5-9 bits de información, un bit de paridad opcional y finaliza con uno o dos bit de parada como se muestra en la Figura 2.15 [29].

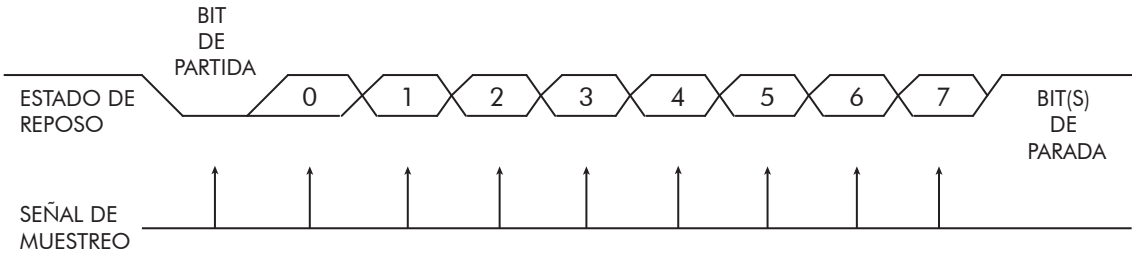


Figura 2.15: Trama de bits para comunicación UART (cambiar)

El protocolo utiliza sólo dos líneas de comunicación y puede ser configurado para ser simplex (El flujo de información sólo tiene un sentido), half duplex (Los dispositivos pueden enviar y recibir información pero se deben turnar) o full duplex (Los dispositivos pueden enviar y recibir información al mismo tiempo). Este protocolo sólo ocupa dos líneas de comunicación.
Los términos que ocupa este protocolo son:

- **RX:** Puerto de recepción del dispositivo.
- **TX:** Puerto de transmisión del dispositivo.

Un diagrama de la comunicación es señalado en la Figura 2.16

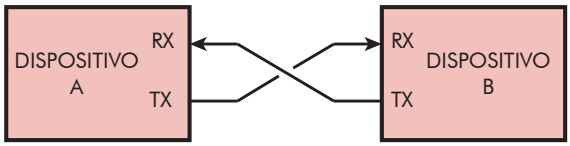


Figura 2.16: Diagrama de comunicación UART

2.3.4. Universal Synchronous/Asynchronous Receiver Transmitter (USART)

Este protocolo es similar a UART, la diferencia radica en que el bus de información es acompañado por la señal de reloj que sincroniza el mensaje. En UART es el dispositvo quien

internamente genera un reloj para sincronizar el mensaje para separar y empaquetar los bits, por lo que se debe conocer el baudrate de antemano.

La transmisión de la señal de reloj permite incrementar la tasa de transmisión llegando a velocidades de 4 Mb/s. USART tiene la capacidad de generar buses más complejos adaptándose a protocolos como IrDA, LIN, RS-485 y MODBUS entre otros. Además USART puede ocuparse de forma asíncrona generando las mismas tramas de comunicación que UART.
La terminología de este protocolo es [30]:

- **RX:** Puerto de recepción del dispositivo.
- **TX:** Puerto de transmisión del dispositivo.
- **CLK:** Línea por la cual se transmite la señal del reloj de sincronización.

Un diagrama de la comunicación es señalado en la Figura 2.17

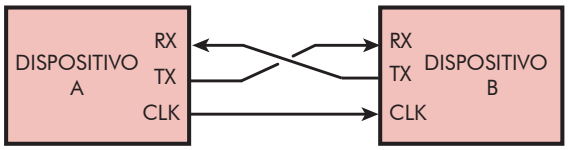


Figura 2.17: Diagrama de comunicación USART

En la Tabla 2.8 se hace un resumen de las principales características de los buses de comunicación serial expuestos en esta sección.

Tabla 2.8: Comparación de buses de comunicación serial

	Bus de comunicación			
Característica	SPI	I2C	UART	USART
Sincronismo	Síncrono	Síncrono	Asíncrono	Síncrono/Asíncrono
N° mínimo de cables	4	2	2	3
Duplex	Full	Half	Full	Full
Velocidad máxima	10-20 Mb/s	3.4 Mb/s - 5 Mb/s	115.2 kb/s - 1Mb/s	4 Mb/s
Relación entre dispositivos	Maestro / Múltiples Esclavos	Múltiples Maestros / Múltiples Esclavos	Punto a punto	Punto a punto

3 Diseño de Tarjeta de Pruebas

En este capítulo se expondrán los elementos y características que comprenden el diseño de la Tarjeta de Pruebas. Se comenzará con el listado de propósitos para los cuales se necesita realizar la tarjeta. Luego se enumerarán las características de los elementos que se incluyen, su interconexión a través de los diagramas de bloques, los esquemáticos correspondientes y la lista de materiales necesarios para su montaje.

3.1. Propósitos

El diseño y uso de la tarjeta de diseño cumple objetivos constructivos dentro de la Memoria y estos son listados a continuación:

- **Uso del software:** El uso previo al diseño de la tarjeta final, permite obtener experiencia y conocer las herramientas que ofrece el software como lo son la creación de 'footprints' para componentes, modelado 3D, realización de esquemáticos y exportación de archivos de diseño; los cuales son elementos básicos para el diseño de las tarjetas electrónicas.
- **Programación de los integrados:** La posibilidad de poder utilizar los integrados permitirá familiarizarse con los lenguaje de programación (C y VHDL), acelerando el proceso de configuración de la tarjeta final sin correr el riesgo de causar daños en el diseño final.
- **Evaluación del rendimiento de los integrados:** Dado que los integrados son los a utilizar en el diseño final, se puede realizar una estimación del desempeño de estos. Pudiendo discriminar y corregir elementos del diseño final a partir de los requisitos propuestos.

3.2. Elección de Integrados

La elección de integrados fue parte del trabajo previo de la tesis, en donde las familias estudiadas en el capítulo anterior fueron evaluadas en base los criterios de la Tabla 2.7, en base esto fue elegido el microcontrolador PIC32MZ y la FPGA MACHXO2 HC600.

El microcontrolador fue elegido dada las siguientes razones:

- Modelo de alta gama de Microchip.
- Posesión del programador en el laboratorio lo que implica una reducción de costos para la memoria.
- Unidad aritmética de punto flotante lo que facilita el procesamiento de datos e implementación del lazo de control.
- Procesador de alta velocidad y ADC interno con una resolución de 12 bits y más de 4 canales independientes.

La FPGA fue elegida por las siguientes razones:

- FPGA de bajo costo y consumo.
- Empaquetado pequeño y fácil montaje (48-QFN).
- Oscilador interno y nivel lógico de 3.3 V compatible con la lógica del microcontrolador.
- Capaz de realizar tareas simples de comunicación y procesamiento de datos.

3.3. Características

La tarjeta de pruebas es capaz de implementar un sistema de control a través la programación y comunicación de sus dispositivos lógicos. La tarjeta contiene la FPGA MACHXO2-640HC y el Microcontrolador PIC32MZEF512. La FPGA cuenta con una memoria Flash no volátil integrada además de funciones integradas para comunicación I2C, SPI, contadores/timers y una gran cantidad de pines i/o de alta velocidad con compatibilidad para distintos niveles lógicos. El microcontrolador de alto rendimiento contiene una memoria Flash interna, bloque DSP de doble precisión para operaciones de punto flotante y periféricos para comunicaciones USB, CAN, SPI e I2C. Ambos integrados contienen un oscilador interno para tareas que no requieran una alta precisión y además se agrega un cristal externo de 50 MHz para la FPGA.

Para la programación de los dispositivos se deben conectar a los puertos ICSP y JTAG, los programadores PICKIT4 y HW-USBN-2B correspondientemente.

La tarjeta de pruebas incluye:

- El microcontrolador PIC32MZ0512EFE100-I PT.
- La FPGA LCMXO2-640HC-4SG48I.
- Puertos 2 x 50 pin I/O de uso general (Microcontrolador PIC32MZ).
- Puertos 2 x 24 pin I/O de uso general (FPGA Lattice MACHXO2).
- Suministro de alimentación de 5 V a través de pines de alimentación.
- Jack USB mini B para alimentación 5V.
- Puerto 6 pin header ICSP para programación (Microcontrolador PIC32MZ.
- Puerto 8 pin header JTAG para programación (FPGA Lattice MACHXO2)

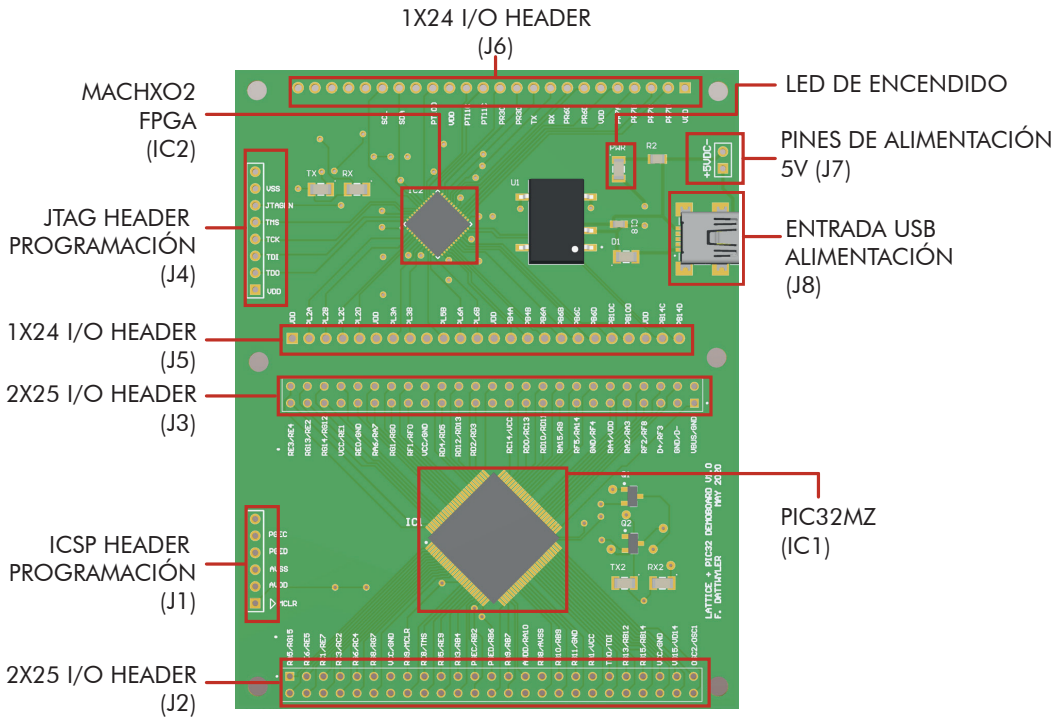


Figura 3.1: Vista superior de la tarjeta de Pruebas

3.4. Diagrama de alimentación

La tarjeta puede ser alimentada desde el Header J7 o el puerto USB con un voltaje de 5 V, luego esta tensión es regulada por el convertidor DC/DC PDS2-S5 quien la reduce a 3.3 V para poder alimentar, el microcontrolador, la FPGA y los programadores conectados. La entrada de voltaje cuenta con un diodo de protección contra inversión de polaridad y un led que indica que la tarjeta se encuentra alimentada.

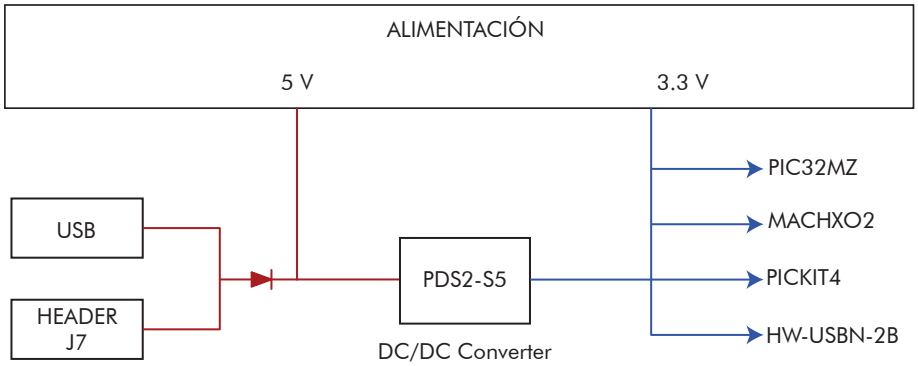


Figura 3.2: Diagrama de alimentación de la Tarjeta de Pruebas

3.5. Diagrama de buses

En la Figura 3.3 se muestra la ubicación en la que se encuentran los buses de los integrados. En las Tablas 3.1 y 3.2 se encuentran las asignaciones para los puertos de programación. En las Tablas 3.3 y 3.4 se encuentran los puertos del microcontrolador PIC32MZ y finalmente en las Tablas 3.5 y 3.6 los puertos de la FPGA MACHXO2-640HC. Para más información acerca de las funciones y programación de los integrados por favor referirse a los manuales y hojas de datos correspondientes.

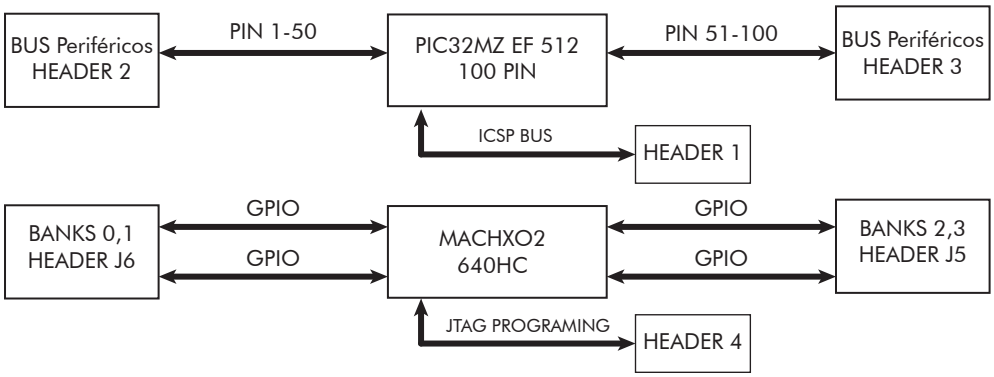


Figura 3.3: Diagrama de distribución de pines de la Tarjeta de Pruebas

Tabla 3.1: Pinout Header J1

n° PIN	Función
1	MCLR
2	VDD
3	VSS
4	PGED
5	PGEC
6	NC

Tabla 3.2: Pinout Header J4

n° PIN	Función
1	VDD
2	TDO
3	TDI
4	TCK
5	TMS
6	JTAGGEN
7	GND
8	NC

Tabla 3.3: Pinout Header J2

n° PIN	Función PIC32	n° PIN	Función PIC32
1	AN23/AERXERR/RG15	26	PGEC2/AN46/RPB6/RB6
2	EBIA5/AN34/PMA5/RA5	27	PGED2/AN47/RPB7/RB7
3	EBID5/AN17/RPE5/PMD5/RE5	28	VREF-/CVREF-/AN27/AERXD2/RA9
4	EBID6/AN16/PMD6/RE6	29	VREF+/CVREF+/AN28/AERXD3/RA10
5	EBID7/AN15/PMD7/RE7	30	AVDD
6	EBIA6/AN22/RPC1/PMA6/RC1	31	AVSS
7	EBIA12/AN21/RPC2/PMA12/RC2	32	EBIA10/AN48/RPB8/PMA10/RB8
8	EBIWE/AN20/RPC3/PMWR/RC3	33	EBIA7/AN49/RPB9/PMA7/RB9
9	EBIOE/AN19/RPC4/PMRD/RC4	34	AN5/RPB10/PMA13/RB10
10	AN14/RPG6/SCK2/RG6	35	AN6/ERXERR/AETXERR/RB11
11	EBIA4/AN13/RPG7/SDA4/PMA4/RG7	36	VSS
12	EBIA3/AN12/C2IND RPG8/SCL4/PMA3/RG8	37	VDD
13	Vss	38	TCK/EBIA19/AN29/RA1
14	VDD	39	TDI/EBIA18/AN30/RPF13/SCK5/RF13
15	MCLR	40	TDO/EBIA17/AN31/RPF12/RF12
16	EBIA2/AN11/C2INC/ERXCLK/EREFCLK/R PG9/PMA2/RG9	41	BIA11/AN7/ERXD0/ 2AECRS/PMA11/RB1
17	TMS/EBIA16/AN24/RA0	42	AN8/ERXD1/AECOL/RB13
18	AN25/AERXD0/RPE8/RE8	43	EBIA1/AN9/ERXD2/AETXD3/RPB14
19	AN26/AERXD1/RPE9/RE9	44	EBIA0/AN10/RPB15/OCFB/RB15
20	AN45/C1INA/RPB5/RB5	45	VSS
21	AN4/C1INB/RB4	46	VDD
22	AN3/C2INA/RPB3/RB3	47	AN32/AETXD0/RPD14/RD14
23	AN2/C2INB/RPB2/RB2	48	AN33/AETXD1/RPD15/SCK6/RD15
24	PGEC1/AN1/RPB1/RB1	49	OSC1/CLKI/RC12
25	PGED1/AN0/RPB0/RB0	50	OSC2/CLKO/RC15

Tabla 3.4: Pinout Header J3

n° PIN	Función PIC32	n° PIN	Función PIC32
51	VBUS	76	RPD1/SCK1/RD1
52	VUSB3V3	77	EBID14/ETXEN/RPD2/PMD14/RD2
53	VSS	78	EBID15/ETXCLK/RPD3/PMD15/RD3
54	D-	79	EBID12/ETXD2/RPD12/PMD12/RD12
55	D+	80	EBID13/ETXD3/PMD13/RD13
56	RPF3/USBID/RF3	81	SQICS0/RPD4/RD4
57	EBIRDY3/RPF2/SDA3/RF2	82	SQICS1/RPD5/RD5
58	EBIRDY2/RPF8/SCL3/RF8	83	VDD
59	EBICS0/SCL2/RA2	84	VSS
60	EBIRDY1/SDA2/RA3	85	EBID11/ETXD1/RPF0/PMD11/RF0
61	EBIA14/PMCS1/PMA14/RA4	86	EBID10/ETXD0/RPF1/PMD10/RF1
62	VDD	87	EBID9/ETXERR/RPG1/PMD9/RG1
63	VSS	88	EBID8/RPG0/PMD8/RG0
64	EBIA9/RPF4/SDA5/PMA9/RF4	89	TRCLK/SQICLK/RA6
65	EBIA8/RPF5/SCL5/PMA8/RF5	90	TRD3/SQID3/RA7
66	AETXCLK/RPA14/SCL1/RA14	91	EBID0/PMD0/RE0
67	AETXEN/RPA15/SDA1/RA15	92	VSS
68	EBIA15/RPD9/PMCS2/PMA15/RD9	93	VDD
69	RPD10/SCK4/RD10	94	EBID1/PMD1/RE1
70	EMDC/AEMDC/RPD11/RD11	95	TRD2/SQID2/RG14
71	EMDIO/AEMDIO/RPD0/RTCC/INT0/RD	96	TRD1/SQID1/RG12
72	SOSCI/RPC13/RC13	97	TRD0/SQID0/RG13
73	SOSCO/RPC14/T1CK/RC14	98	EBID2/PMD2/RE2
74	VDD	99	EBID3/RPE3/PMD3/RE3
75	VSS	100	EBID4/AN18/PMD4/RE4

Tabla 3.5: Pinout Header J5

n° PIN	PIN MACHXO2
1	VDD
2	PL2A
3	PL2B
4	PL2C
5	PL2D
6	VDD
7	PL3A
8	PL3B
9	NC
10	PL5B
11	PL6A
12	PL6B
13	VDD
14	PB4A
15	PB4B
16	PB6A
17	PB6B
18	PB6C
19	PB6D
20	PB10C
21	PB10D
22	VDD
23	PB14C
24	PB14D

Tabla 3.6: Pinout Header J6

n° PIN	**PIN MACHXO2**
1	VDD
2	PR7D
3	PR7C
4	PR7B
5	PR7A
6	VDD
7	PR6D
8	PR6C
9	RX
10	TX
11	PR3D
12	PR3C
13	PT11D
14	PT11C
15	VDD
16	PT10D
17	NC
18	SDA
19	SCL
20	NC
21	NC
22	NC
23	NC
24	NC

3.6. Esquemáticos

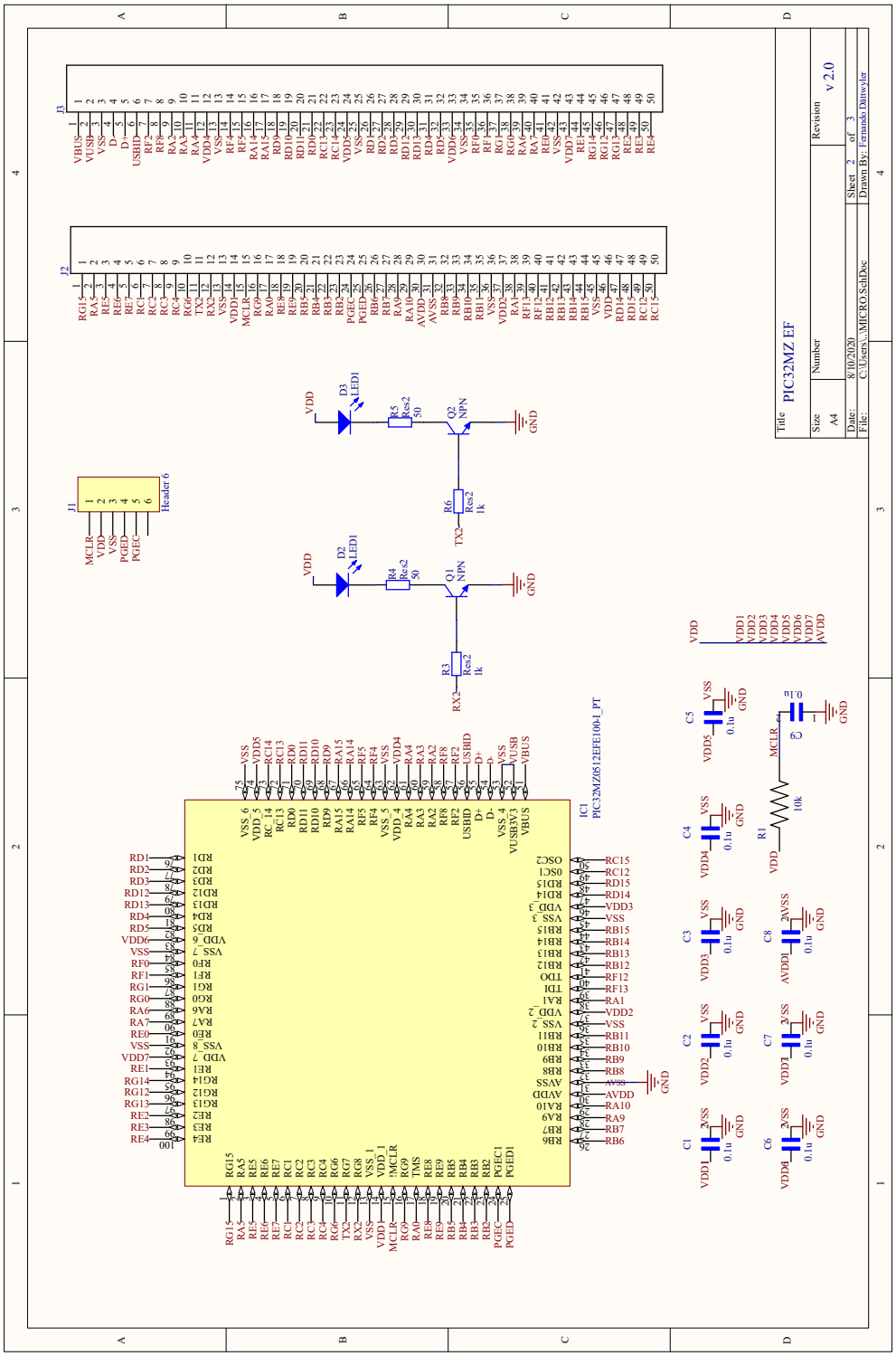


Figura 3.4: Esquemático del microcontrolador PIC32

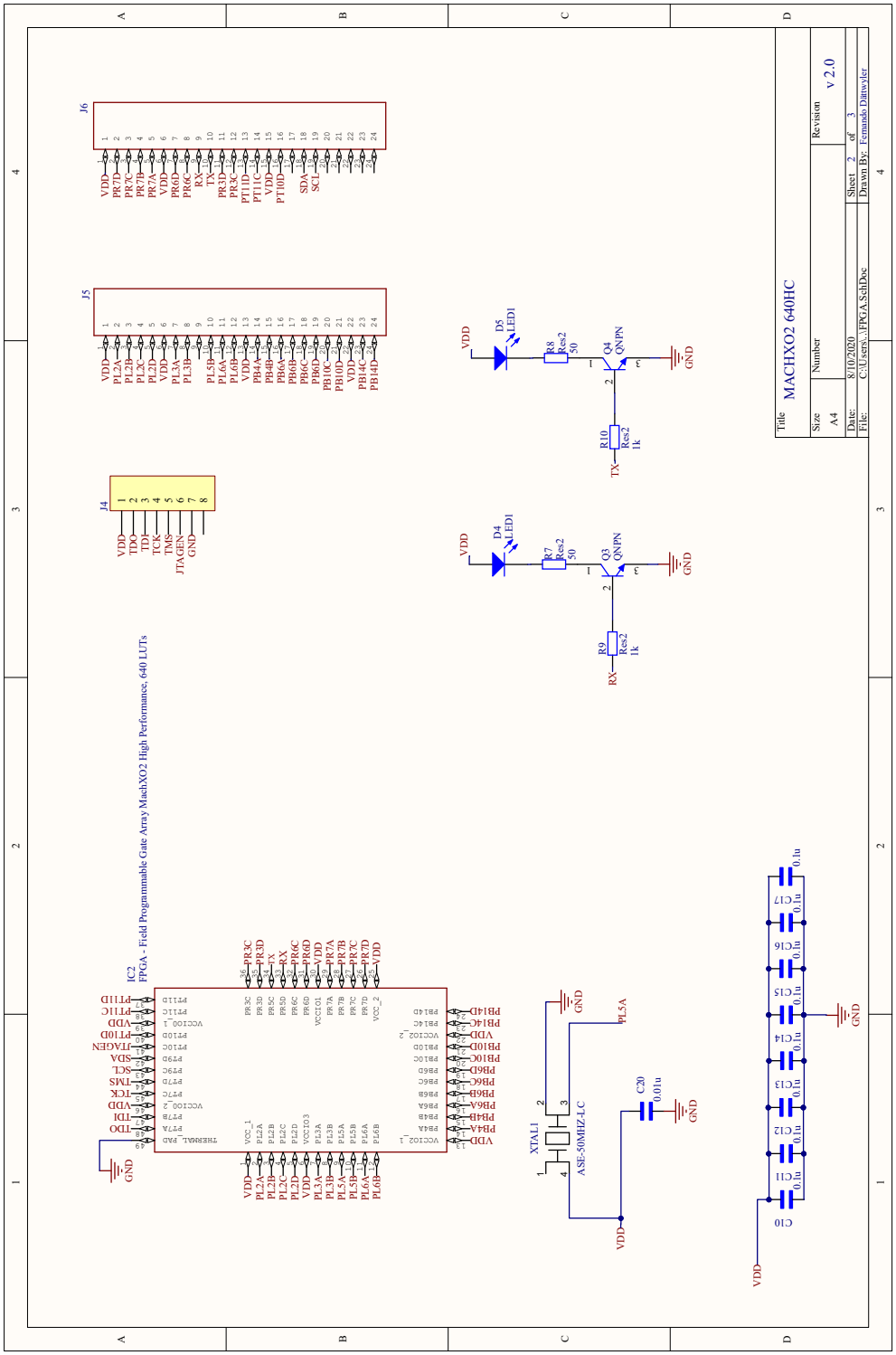


Figura 3.5: Esquemático de la FPGA MACHX02

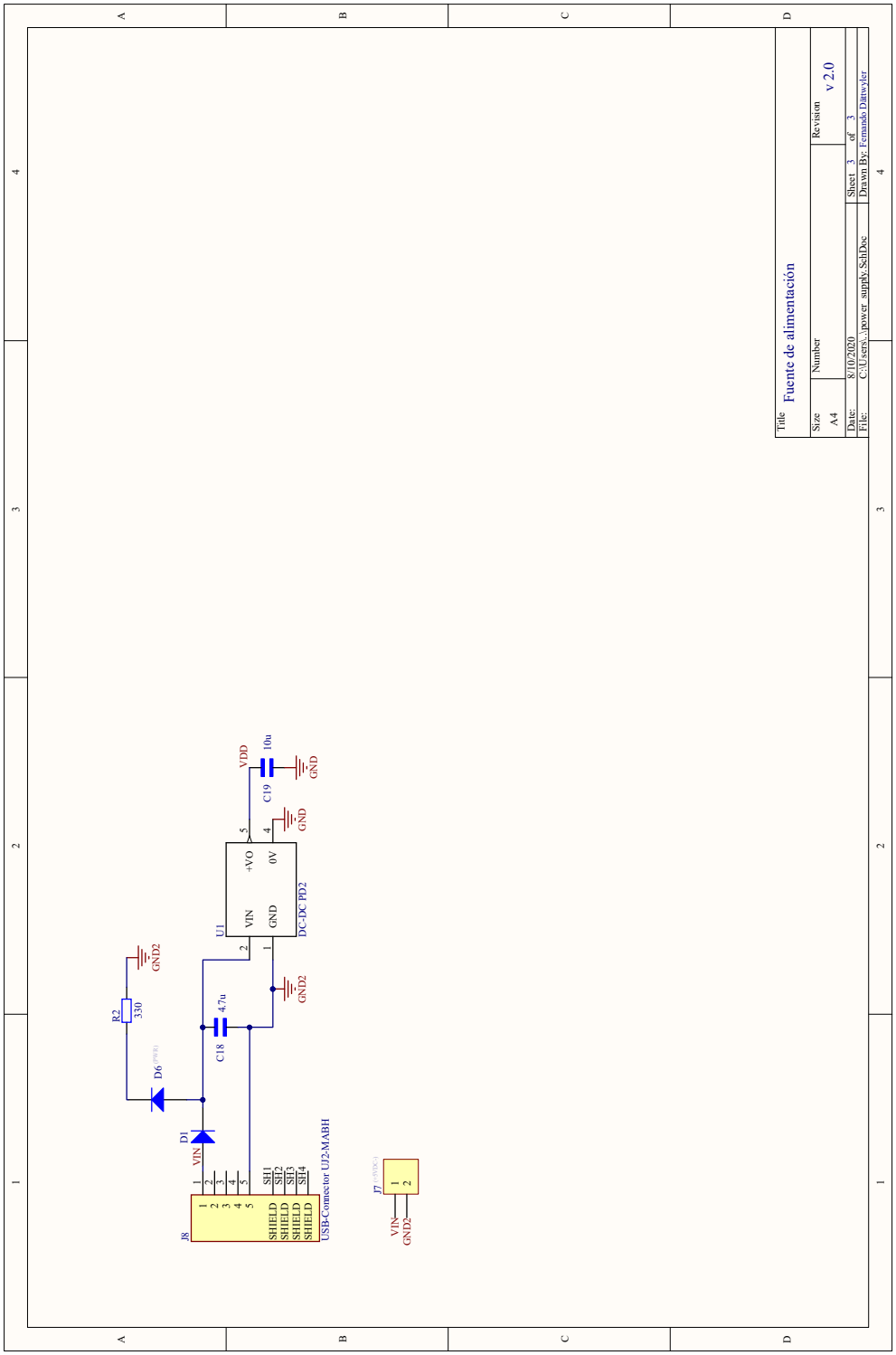


Figura 3.6: Esquemático de la fuente de alimentación

3.7. Lista de materiales

En la Tabla 3.7 se muestra la lista de materiales para la Tarjeta de Pruebas

Tabla 3.7: Lista de materiales Tarjeta de Pruebas

Item #	**Etiqueta**	**N° de pieza del fabricante**	**Cantidad**
1	C1, C2, C3, C4, C5, C6, C7,C8, C9, C10, C11, C12, C13, C14, C15, C16, C17	C0603C104K8RACTU	17
2	C18	C0603C475K8PACTU	1
3	C19	C0603C106M7PAC7867	1
4	C20	C0603C475K8PACTU	1
5	D1	C0603C104K8RACTU	1
6	D2, D3, D4, D5,D6	BL-HB333Q-AV-TRB	5
7	IC1	PIC32MZ0512EFE100-I_PT	1
8	IC2	LCMXO2-640HC-4SG48C	1
9	J1	68000-406HLF	1
10	J2,J3	2-825433-5	4
11	J4	68002-408HLF	1
12	J5,J6	0022284245	2
13	J7	68000-406HLF	1
14	J8	UJ2-MBH-1-SMT-TR	1
15	Q1, Q2,Q3,Q4	ERJ-P06J200V	4
16	R1	RC0805JR-0710KL	1
17	R2	RC0805JR-07330RL	1
18	R3,R6,R9,R10	RT0805BRB071KL	4
19	R4,R5,R7,R8	RC0805FR-0754R9L	4
20	U1	PDS2-S5-S3-M-TR DC DC	1
21	XTAL1	ASE-50.000MHZ-LC-T	1

3.8. Dimensiones y vistas

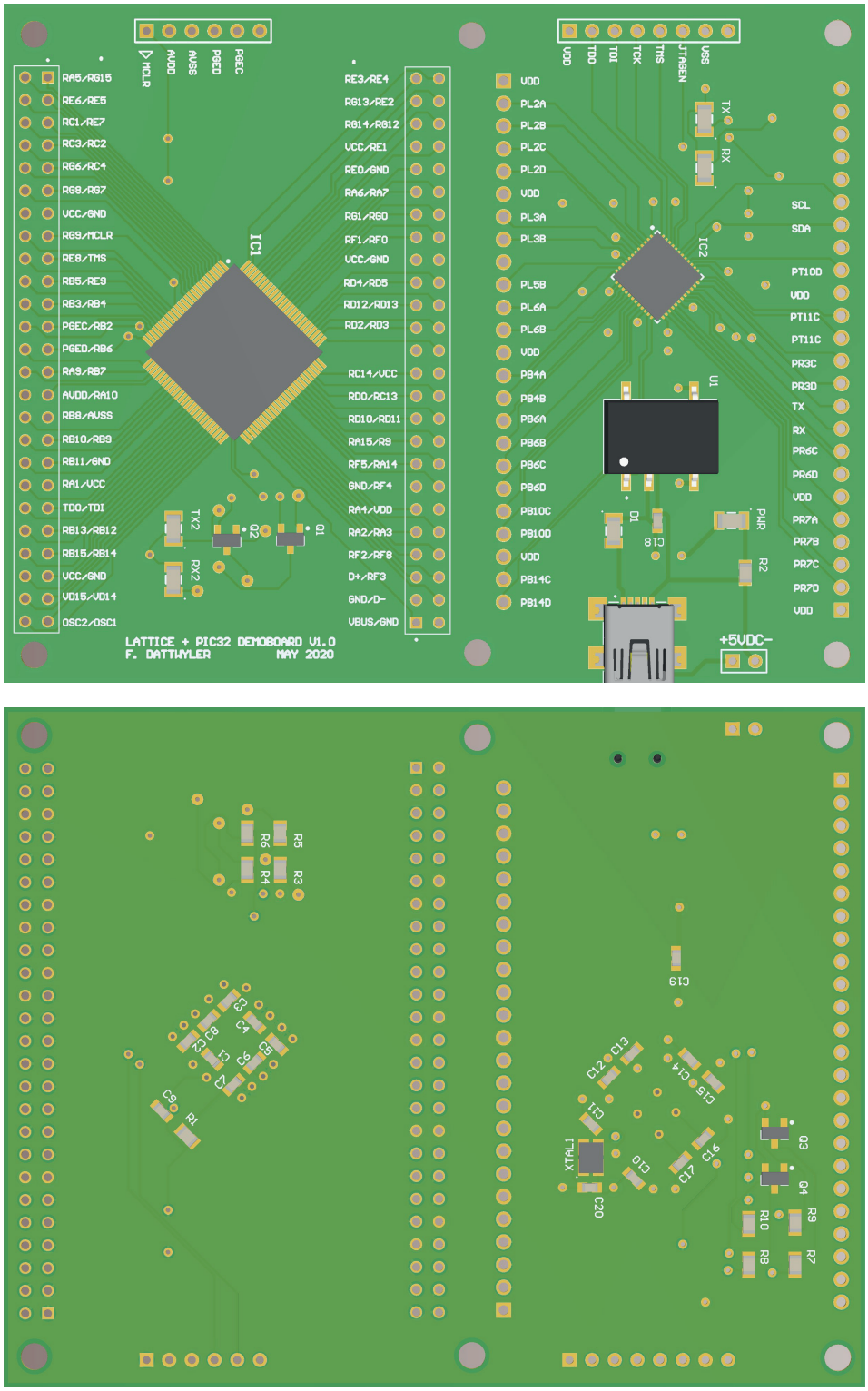


Figura 3.7: Vistas superior e inferior de la tarjeta de Pruebas

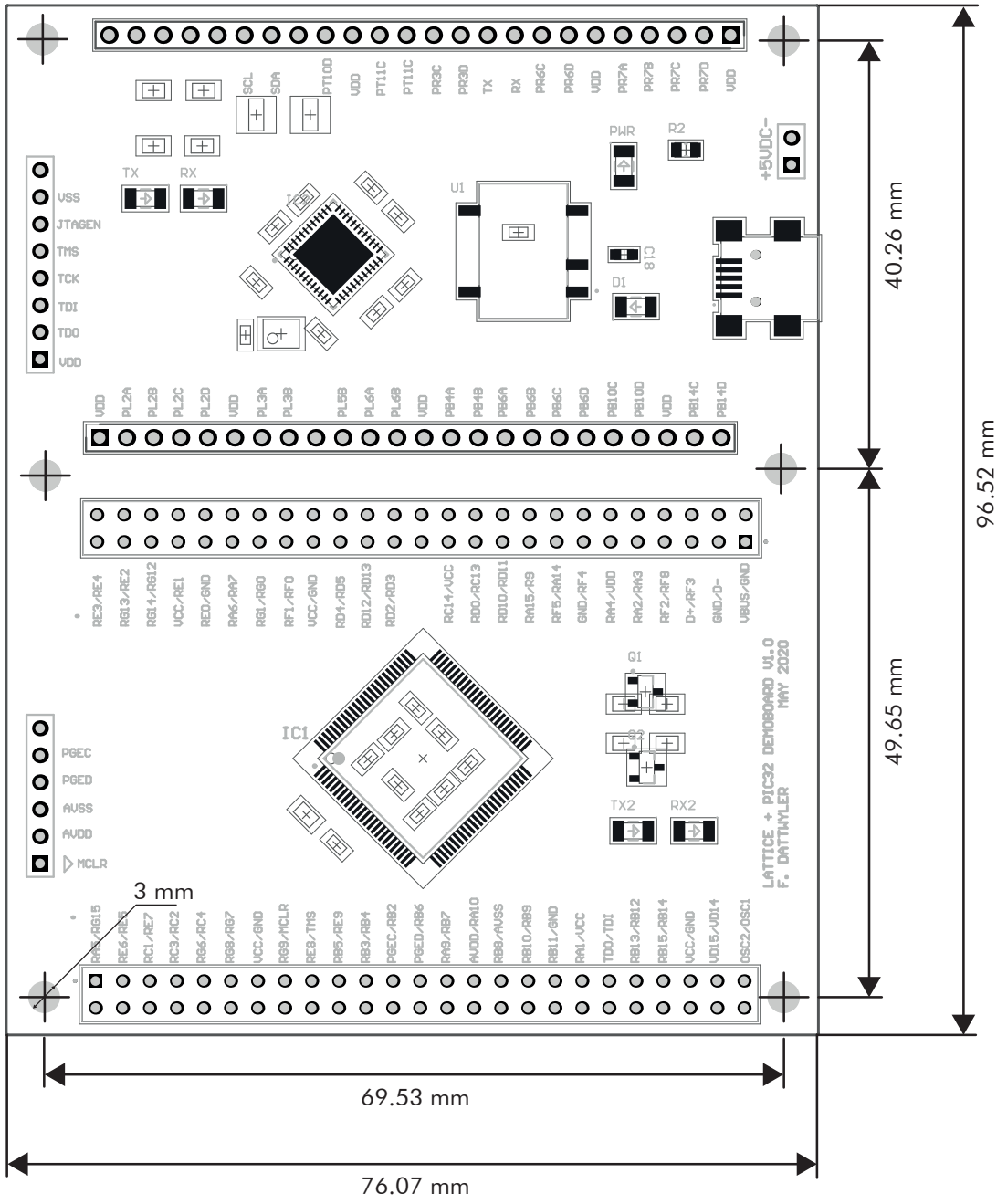


Figura 3.8: Dimensiones físicas de la tarjeta de Pruebas

4 Diseño Lógico y Señales de la Tarjeta de Control

En este capítulo se detallará la estructura lógica y funcional que se implementará en la Tarjeta de Control. El capítulo comenzará con la descripción de los elementos del sistema. Luego se detallarán los requisitos de funcionamiento, a partir de estos se propondrá una solución para la cual se explicará el modo de funcionamiento y los bloques involucrados para su implementación.

4.1. Descripción del sistema

El sistema de estudio como se observa en la Figura 2.3 se divide en cuatro elementos principales, los cuales son:

- **Hardware de potencia:** Corresponde a la celda del módulo SST, la cual se desea controlar. Esta se compone de dos puentes H externos, que conectan la tarjeta con la red y dos puentes H internos, que conforman el puente activo (DAB) que permite la bidireccionalidad de la potencia, el control del voltaje y la corriente circulante. En la celda también son implementados los tiempos muertos correspondientes a cada semiconductor de forma análoga con componentes discretos.
- **Adquisidor de datos:** Corresponde al conversor análogo digital (ADC), que obtiene las mediciones de la corriente del primario y los voltajes de los condensadores de entrada y salida del DAB. Las señales son codificadas en palabras de 16 bits, las que son transmitidas al sistema de control de celda.
- **Sistema de control de celda:** Es el sistema en el cual se implementa el lazo de control. Este adquiere las referencias que provienen del ADC y del sistema de control externo. Este se encarga de generar los pulsos de disparos que se dirigen al hardware de potencia.
- **Sistema de control externo:** Es el sistema que se encarga del control del convertidor, este entrega las referencias de los voltajes de los condensadores de entrada y salida, la referencia de la corriente, el modo de operación del DAB y los pulsos de disparo para los puentes H externos.

Los elementos anteriormente descritos se relacionan entre sí a través de señales, estas son mostradas en la Figura 4.1 y detalladas a continuación:

- $SM1_{1,2,3,4}$: Señales correspondientes a los pulsos de disparos que se dirigen al puente H SM1.
- $SM2_{1,2,3,4}$: Señales correspondientes a los pulsos de disparos que se dirigen al puente H SM2.
- $SP_{1,3,ENB}$: Señales correspondientes a los pulsos de disparos que se dirigen al puente H SP y su señal de ENABLE.

- $SS_{1,3,ENB}$: Señales correspondientes a los pulsos de disparos que se dirigen al puente H SS y su señal de ENABLE.
- V_{C1} : Medición de voltaje del condesador C1.
- V_{C2} : Medición de voltaje del condesador C2.
- i_p : Corriente del primario correspondiente al DAB.
- SDO_1 : Señal cuantificada del voltaje del condesador C1.
- SDO_2 : Señal cuantificada del voltaje del condesador C2.
- SDO_3 : Señal cuantificada de la corriente i_p.
- CLK_{out} : Señal de reloj de transmisión del ADC.
- SCK : Señal de reloj para la captura de datos.
- CNV : Bandera que habilita el reloj de conversión y transmisión de datos.

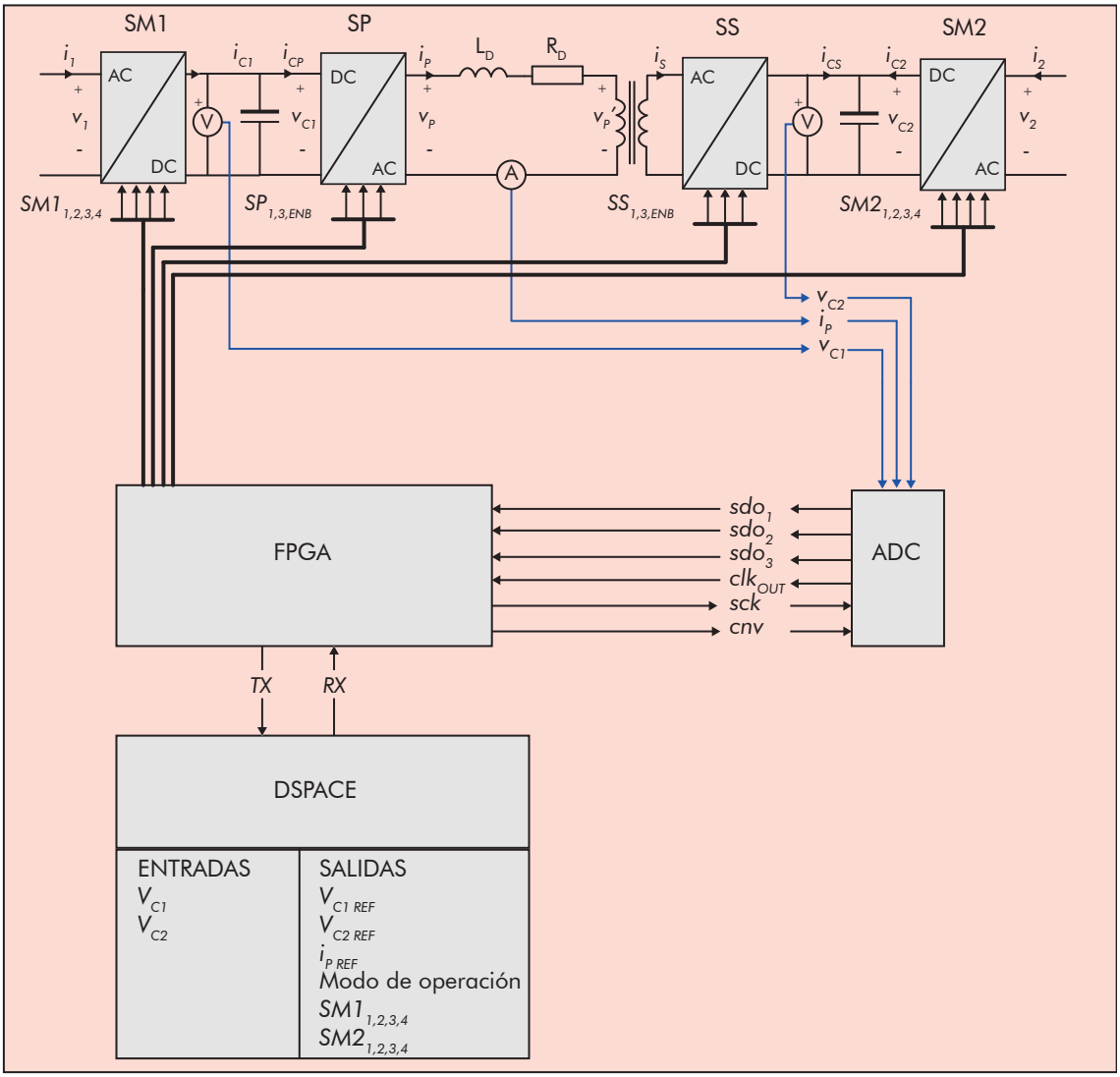


Figura 4.1: Diagrama del Sistema de estudio

4.2. Requisitos de diseño

Los requisitos de diseño a cumplir son:

- La implementación debe adquirir los datos provenientes desde los sensores con una resolución de 12 bits con un periodo de muestreo mínimo de 1 μs.
- Se debe incluir una unidad que pueda responder ante las señales de fallas y que comande el funcionamiento del control de celda.
- Las banderas de sobre voltaje y sobre corriente deben ser accionadas cuando se recibe una medición que supere el 80 % del valor máximo indicado por el fabricante.
- Se debe disponer de una interfaz que permita adquirir las mediciones provenientes de los sensores.
- Se debe incluir un módulo de comunicación serial de 8 bits, que comunique el sistema de control de celda con el control externo. El baudrate de la comunicación debe ser de al menos 10 MHz.
- Se debe incluir un módulo que interprete los mensajes recibidos y codifique los mensajes transmitidos.
- La implementación debe generar los pulsos de disparos correspondientes al DAB y transmitir los pulsos de disparos de los puentes H externos proveniente del control externo.
- Las tareas de captura de datos, obtención de referencias y cálculo de los disparos a enviar deben realizarce dentro de 10 μs para cumplir el ancho de banda del controlador.

4.3. Funcionamiento

El Sistema de Control de Celda es implementado en una FPGA con el fin de obtener un mayor rendimiento al realizar tareas en paralelo. A pesar de que los módulos pueden operar simultáneamente, para la realización de ciertas tareas y la obtención de nuevos valores de referencias se necesita una secuencialización de las operaciones. Por lo que en esta sección se explicará el modo de funcionamiento del Sistema de Control de Celda.

El Sistema se puede encontrar en tres estados IDLE, OPERACIÓN o EMERGENCIA. El primer estado, corresponde al estado por defecto luego de que la tarjeta es encendida, por lo que es necesario que se realice una secuencia de Inicialización. Durante esta, el sistema de control recibe la información del modo de operación, referencias de voltaje y corriente mientras se estabiliza el voltaje de los condensadores y pasa el estado transitorio del sistema de potencia. En paralelo se envían las mediciones que son adquiridas por el ADC e interpretadas por el módulo de interfaz al sistema de control externo para el cálculo de los puentes externos SM1 y SM2 como es mostrado en el diagrama de tiempo de la Figura 4.2 y cuyas mediciones se encuentran en la Tabla 6.4.

Posterior a la secuencia de inicialización y se procede al estado de operación, en donde habilitan los disparos al puente DAB cerrando el lazo de control, en la sección Trabajos Futuros se hace una nota sobre el efecto de este proceso en el control. Como se ha mencionado anteriormente los módulos trabajan de forma simultánea, por lo que mientras se envían los disparos al puente DAB también se están recibiendo las mediciones provenientes del ADC y transmitiendo datos al Sistema de Control Externo, entre otras tareas. Sin embargo para la obtención de un pulso de disparo con diferente ciclo de trabajo, fase o estado de encendido por causa de un cambio de referencia o perturbación del Sistema de Potencia se genera un camino de datos secuencial.

En consideración a la situación anterior, es necesario recibir un nuevo bus de mediciones provenientes del ADC. En el módulo de Interfaz las mediciones son recepcionadas y cifradas para el

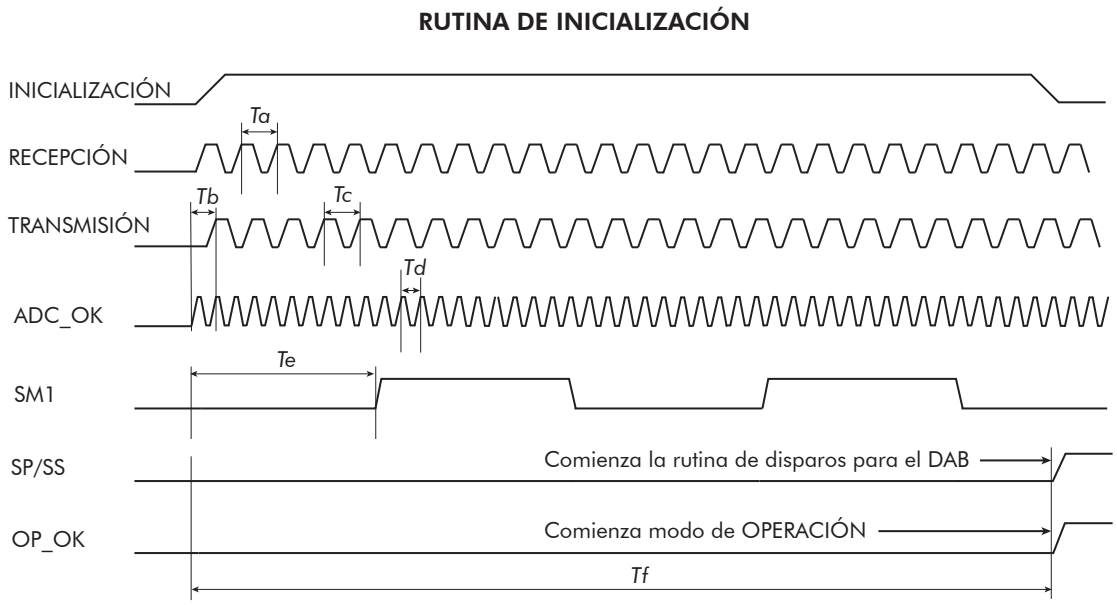


Figura 4.2: Diagrama de tiempo para la operación de Inicialización

envío al Sistema de Control Externo, a través del módulo Codificador y UART correspondientemente, esto se refleja en la Figura 4.3.
Luego de esto, las nuevas referencias de voltaje y corriente son calculadas en el Sistema de Control Externo, estas son recibidas en cuatro mensajes consecutivos en el módulo UART. Los mensajes son Decodificados enviados a los módulos correspondientes, tal como se muestra en la Figura 4.4. Posteriormente se actualiza el estado de las salidas que se representan los disparos de los puentes SM1 y SM2, en paralelo se realiza el cálculo de los ciclos de trabajo y fase para los moduladores esto es ilustrado en las Figuras 4.5 y 4.6 . Finalmente los nuevos pulsos que se diregen al DAB son enviados lo que se muestra en la Figura 4.7.

Este es el camino crítico y representa el tiempo o retardo que genera el sistema de control desde la entrada de la referencia a la salida del disparo del DAB, valor que debe ser tomado en cuenta para la síntesis del controlador. Los valores se encuentran detallados en la Tabla 6.5.

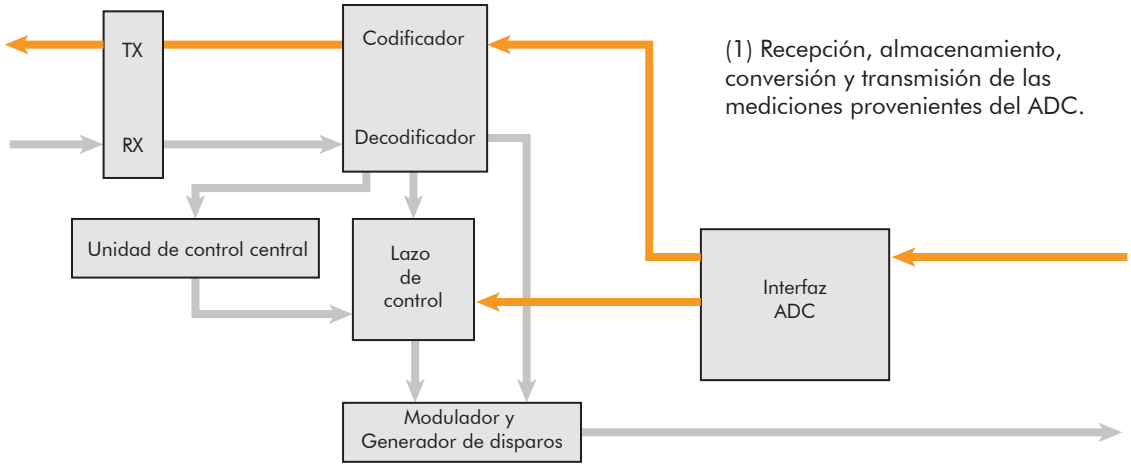


Figura 4.3: Adquisición y transmisión de mediciones hacia el Sistema de Control Externo

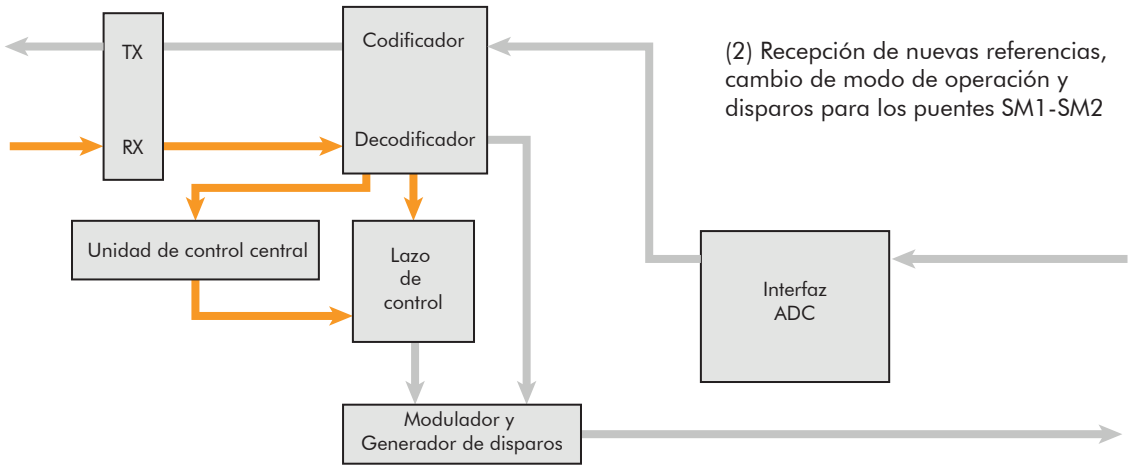


Figura 4.4: Recepción y decodificación de mensajes provenientes del Sistema de Control Externo

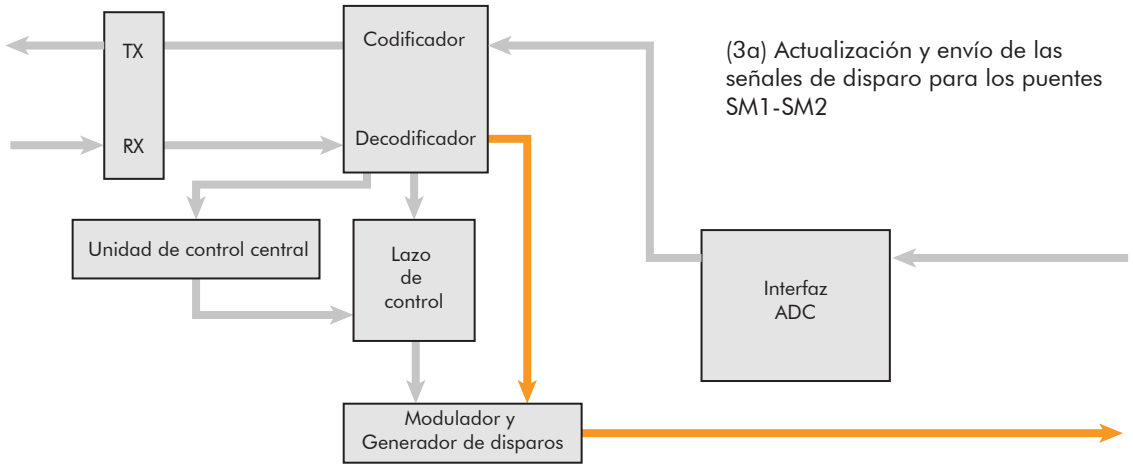


Figura 4.5: Envío de pulsos a los puentes SM1 y SM2

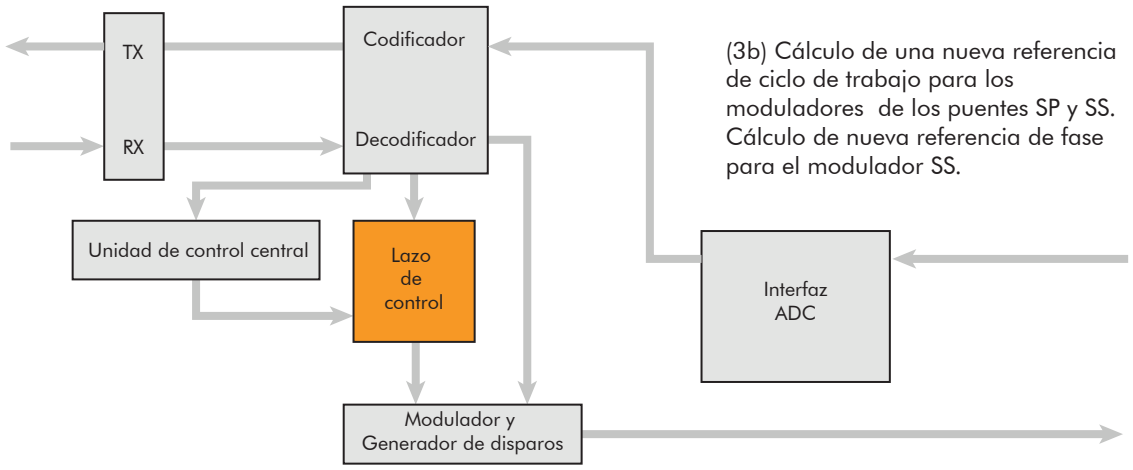


Figura 4.6: Cálculo de nuevas salidas del Lazo de Control

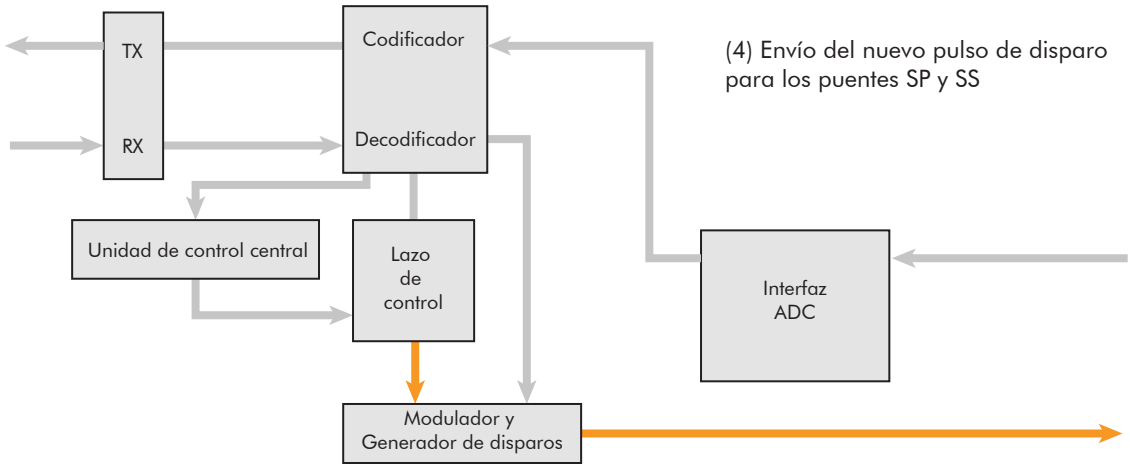


Figura 4.7: Envío de pulsos a los puentes SP y SS

4.4. Implementación

El sistema lógico que lleva a cabo las tareas mencionadas en la sección anterior, se compone de módulos ordenados de forma jerárquica. Estos se relacionan a través de señales que pueden controlar, coordinar y comunicar tanto el sistema de control de celda con el externo. El sistema propuesto se muestra en la Figura 4.8 y los módulos son detallados a continuación.

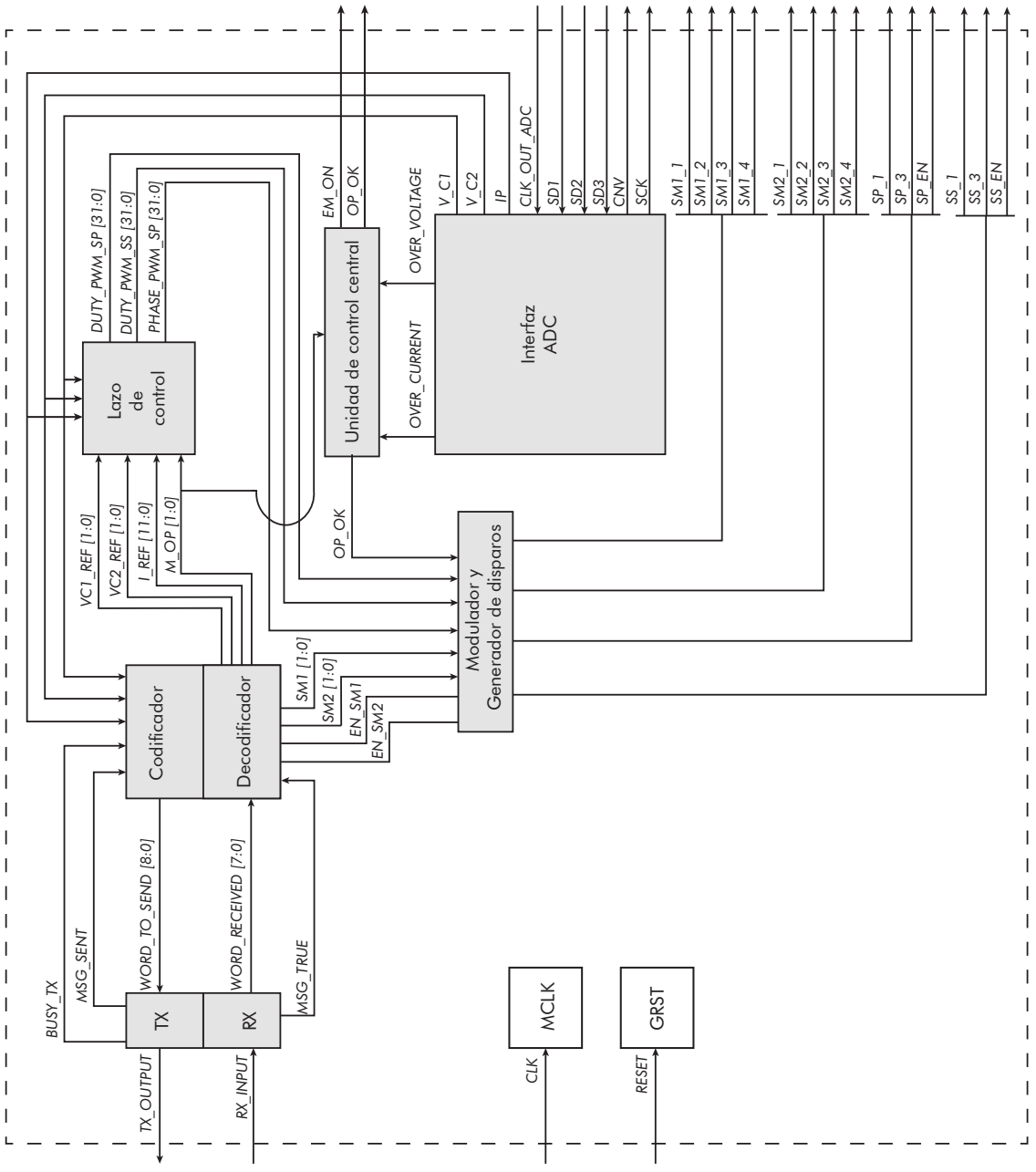


Figura 4.8: Diagrama de bloques del sistema lógico

4.4.1. Unidad de Control Central

En este módulo se programan las rutinas de inicialización y acciones ante fallas. El módulo de se compone de una máquina de estados y tres procesos auxiliares que calculan la lógica de transición de la máquina de estados. En el estado IDLE se permite la operación de la comunicación serial, la adquisición de datos y el lazo de control pero se deshabilita el envío de disparos a los puentes H SP y SS. En estado OPERACION se habilitan el envío de disparos y en el estado de EMERGENCIA enciende los leds indicadores de falla por sobre voltaje o corriente y deshabilitan los disparos que se dirigen al DAB. Para salir del estado de emergencia, la bandera de sobrecorriente y sobrevoltaje deben estar en '0' y el modo de operación debe ser '11', o recibir una señal de RESET.

Las señales de **entrada** son:

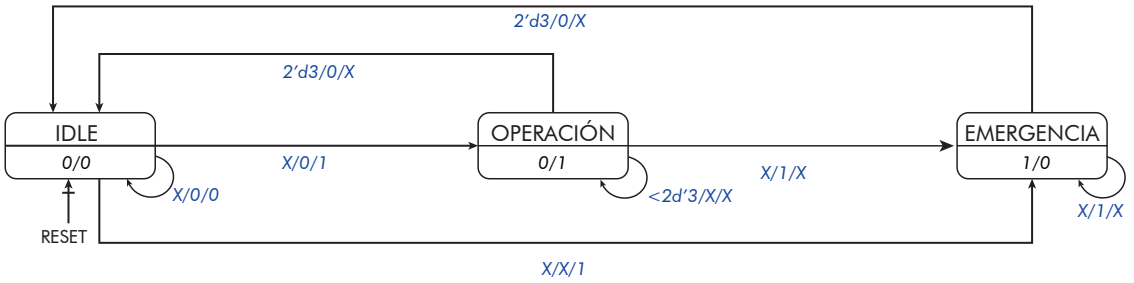
- CLK: Reloj principal de la FPGA.
- RESET: Reset general.
- M_OP [1:0]: Modo de operación.
- OVER_VOLTAGE: Bandera de emergencia de sobre voltaje.
- OVER_CURRENT: Bandera de emergencia de sobre corriente.

Las señales de **salida** son:

- OP_OK: Señal que indica que se encuentra en el modo de OPERACION.
- EM_ON: Señal que indica que se encuentra en el modo de EMERGENCIA.

Las señales **internas** son:

- ESTADOS (IDLE,OPERACION,EMERGENCIA): Definición de estados.
- EMERGENCY_FLG: Bandera que detecta una falla de sobre voltaje o sobre corriente.
- INI_TIME: Bandera que indica el término del temporizador de inicialización.
- INI_OK: Bandera que indica el término de la rutina de inicialización.



ENTRADAS	SALIDAS
M_OP [1:0]	EM_ON
EMERGENCY_FLG	OP_OK
INI_OK	

Figura 4.9: Maquina de estados Unidad de Control Central

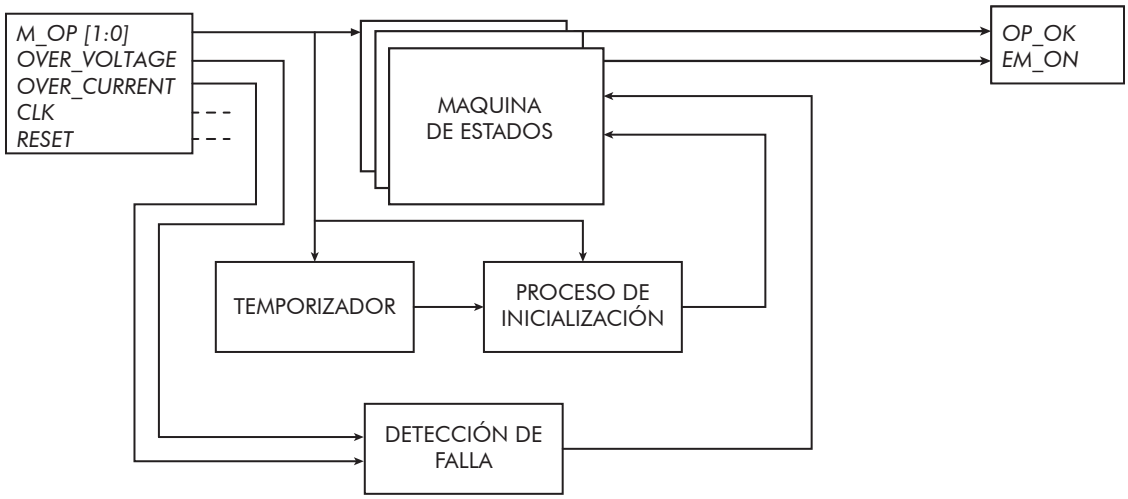


Figura 4.10: Diagrama de la Unidad de Control Central

4.4.2. Módulo UART

Este módulo se encarga de realizar la comunicación entre la FPGA y la DSPACE, envía la información de los voltajes de los condensadores del DAB que han sido previamente codificados. Este módulo se encuentra a su vez dividido en la transmisión y recepción, los cuales son independientes entre sí y cuentan con sus propios generadores de reloj de comunicación.
La transmisión se conforma de una máquina de doce estados que forman una estructura secuencial. El primer estado corresponde al estado IDLE o de reposo, este corresponde al estado por defecto cuando no se quiere realizar una transmisión, luego de una inicialización o de un reset.
Luego de que haya llegado una bandera de transmisión, se haya sincronizado el reloj de comunicación y se haya cumplido la rutina de inicialización se pasa al estado de FB, el cual transmite el bit de partida de la comunicación el cual por protocolo es '0' y se levanta la bandera que indica que el transmisor está ocupado.
Después le prosiguen ocho estados correspondientes a cada bit del mensaje de comunicación comenzando por el bit menos significativo tal como se muestra en la Figura 2.15.
Posteriormente le sigue el estado de STOP que finaliza la comunicación y transmite el bit de parada y el estado STOP2 que indica que el mensaje se ha transmitido satisfactoriamente, limpia los registros y se transiciona al estado inicial IDLE.
El diagrama de estados correspondiente a la máquina de la transmisión se encuentra en la Figura 4.12

Tanto la recepción como la transmisión son sintetizadas como componentes, en donde deben ser definidas sus entradas y salidas para la interconexión con otras componentes y el módulo principal. Las señales son listadas a continuación.

Las señales de **entrada** son:

- CLK: Reloj principal de la FPGA.
- RESET: Reset general.
- TX_FLAG: Bandera de requerir transmisión.
- WORD_TO_SEND [8:0]: Palabra a transmitir por el canal.

Las señales de **salida** son:

- OUTPUT: Bit que se transmite por el canal TX.
- MSG_SENT: Bandera que indica el término de la transmisión.
- BUSY: Bandera que indica que el módulo se encuentra ocupado.

Las señales **internas** son:

- ESTADOS (IDLE, FB, S1, S2, S3, S4, S5, S6, S7, S8,STOP,STOP2): Definición de los estados de la máquina.
- WRITE: Bandera para escribir el bit del mensaje.

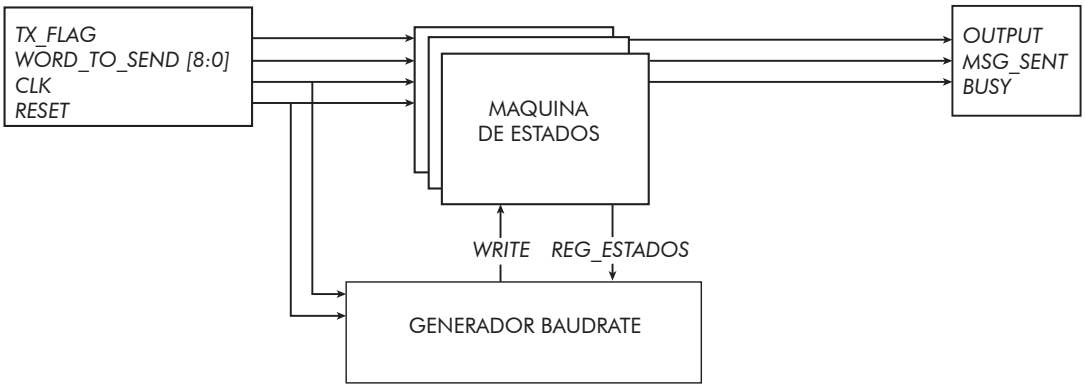


Figura 4.11: Diagrama del transmisor

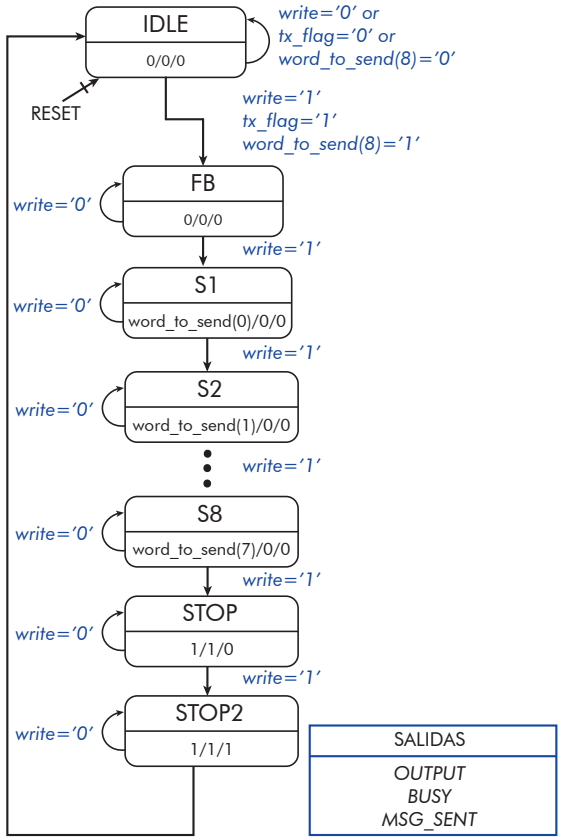


Figura 4.12: Máquina de estados para el módulo de transmisión

La recepción de forma similar a la transmisión, se conforma de una máquina de cuatro estados. El primer estado IDLE, corresponde al estado por defecto, cuando no se está recibiendo un mensaje o luego de una reinicialización. En este se limpian todos lo registros de la máquina de estados. Luego, si se detecta un '0' en el canal de recepción se transiciona al estado FB en el cual se levanta la bandera que indica que el receptor se encuentra ocupado y se sincroniza el reloj de comunicación. En el estado S1 se rellena el bus de 8 bit que se recibe del canal de recepción una vez terminado se transiciona al estado STOP en donde se comprueba que se recibe el bit de parada, se escribe la palabra recibida en el registro correspondiente y se pasa al estado IDLE.

Las señales de **entrada** son:

- CLK: Reloj principal de la FPGA.
- RESET: Reset general.
- INPUT: Señal de entrada del canal RX.

Las señales de **salida** son:

- WORD_RECEIVED: Bit que se transmite por el canal TX.
- BUSY_RX: Bandera que indica el término de la transmisión.
- MSG_TRUE: Bandera que indica que el módulo se encuentra ocupado.

Las señales **internas** son:

- ESTADOS (IDLE,FB,S1,STOP): Definición de estados.
- CONTADOR_BIT [3:0]: Señal que indica el bit a escribir en el registro N_STORED_WORD.
- N_CONTADOR_BIT [3:0]: Señal buffer que que actualiza el valor de CONTADOR_BIT.
- RX_FLAG: Bandera que indica que el canal RX está siendo ocupado.
- STORED_WORD: Registro que almacena los bits capturados del canal RX.
- N_STORED_WORD: Señal buffer que que actualiza el valor de STORED_WORD.
- READ: Bandera que ordena leer el canal RX.

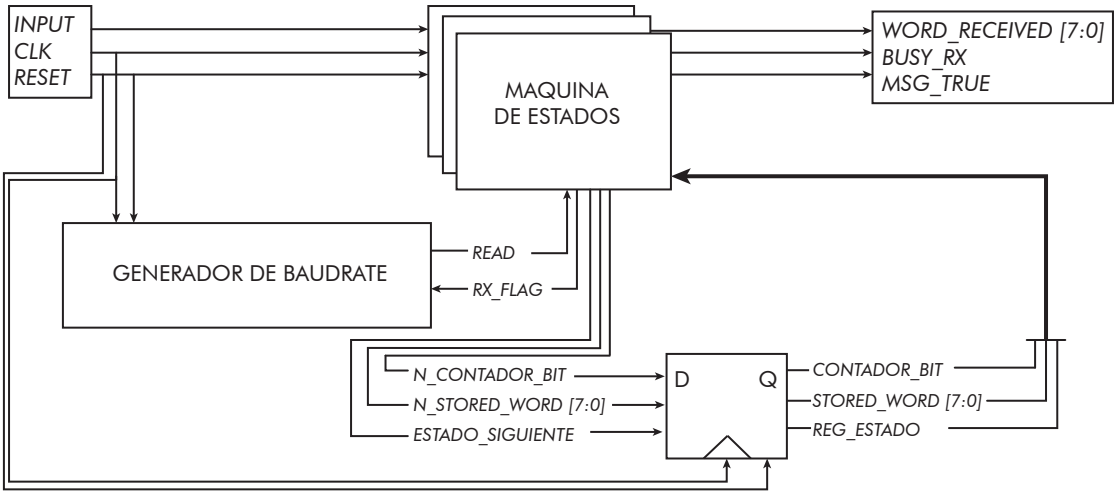


Figura 4.13: Diagrama del receptor

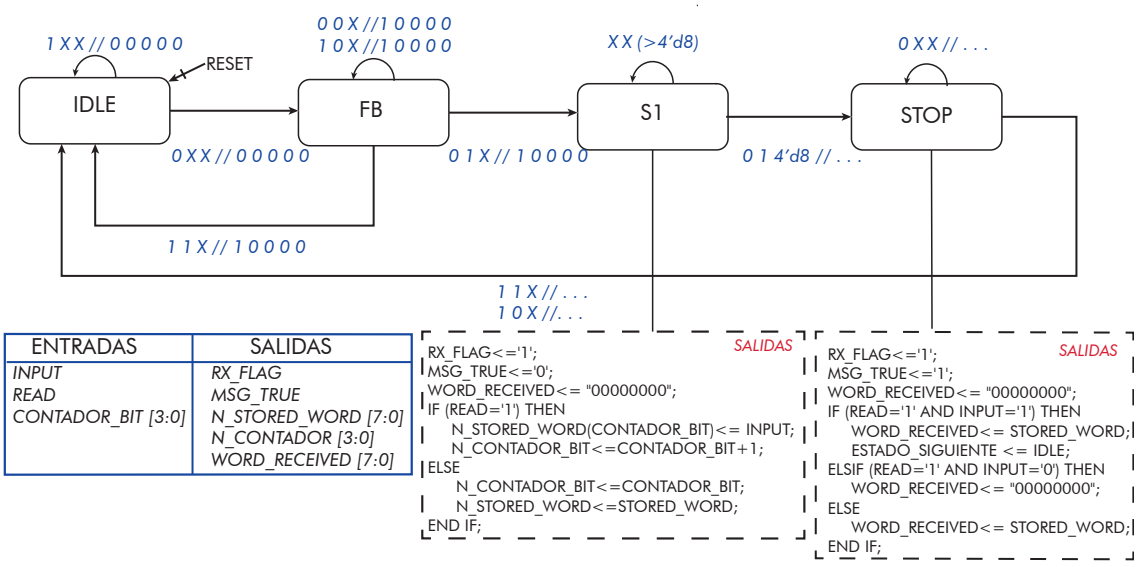


Figura 4.14: Máquina de estados para el módulo de recepción

4.4.3. Codificador y Decodificador

En este módulo se interpretan y codifican los mensajes que son transmitidos y recibidos por el puerto serial. Los mensajes son cifrados en una forma preestablecida en donde los dos bits más significativos representan el tipo de mensaje y el resto la información de los registros objetivos como lo muestra la Figura 4.18.

La estructura del codificador se compone de una máquina de estados la cual envía secuencialmente los buses de mensajes a transmitir por el módulo UART y un proceso que codifica los buses de mediciones que son obtenidos desde el ADC. La máquina cuenta con cinco estados, el primero corresponde al estado por defecto IDLE en el cual se levanta la bandera para la codificación de los mensajes y se espera el término de recepción de los datos en el ADC para pasar al conjunto de estados de transmisión MSG1, MSG2, MSG3 y MSG4, los que tienen como señal de control el estado del módulo de transmisión. Terminada la secuencia de mensajes se vuelve al estado IDLE.

Las señales de **entrada** son:

- CLK: Reloj principal de la FPGA.
- RESET: Reset general.
- BUSY_TX: Bandera que indica que el canal TX está ocupado.
- ADC_OK: Señal que indica la recepción de datos provenientes del ADC.
- V_C1 [11:0]: Medición de voltaje del condensador V_{C1}.
- V_C2 [11:0]: Medición de voltaje del condensador V_{C2}.
- MSG_SENT: Bandera que indica el término de la transmisión de un mensaje.

Las señales de **salida** son:

- WORD_TO_SEND [8:0]: Palabra a enviar por el módulo de transmisión.

Las señales **internas** son:

- ESTADOS_T (IDLE,MSG1,MSG2,MSG3,MSG4): Definición de estados.

- ESTADO_ACTUAL,ESTADO_SIGUIENTE:ESTADOS_T: Declaración del vector de estados.
- V_C1_PT1 [7:0]: Registro que representa los seis bits menos significativos de la medición de V_{C1}.
- V_C1_PT2 [7:0]: Registro que representa los seis bits más significativos de la medición de V_{C1}.
- V_C2_PT1 [7:0]: Registro que representa los seis bits menos significativos de la medición de V_{C2}.
- V_C2_PT2 [7:0]: Registro que representa los seis bits más significativos de la medición de V_{C2}.
- V_C1_PT1_B [5:0]: Señal buffer que que actualiza el valor de V_C1_PT1.
- V_C1_PT2_B [5:0]: Señal buffer que que actualiza el valor de V_C1_PT2.
- V_C2_PT1_B [5:0]: Señal buffer que que actualiza el valor de V_C2_PT1.
- V_C2_PT2_B [5:0]: Señal buffer que que actualiza el valor de V_C2_PT2.

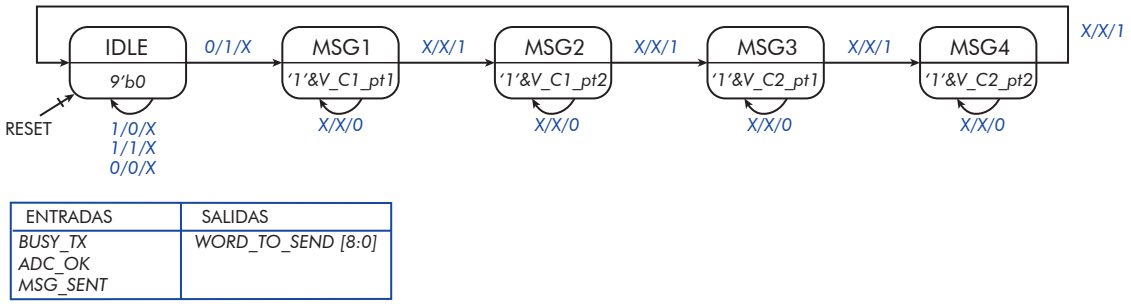


Figura 4.15: Máquina de estados para el módulo de codificación

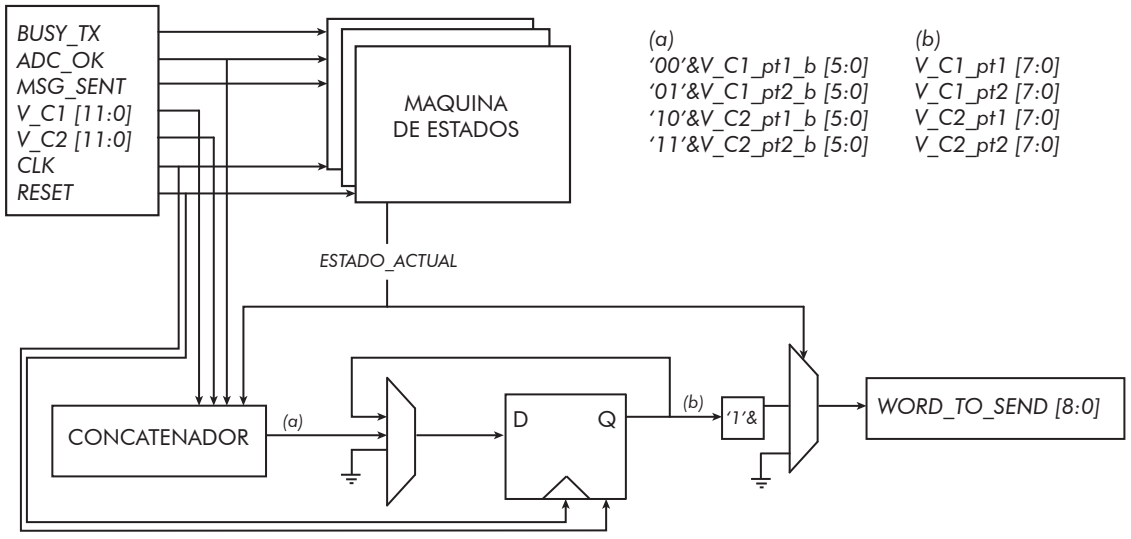


Figura 4.16: Máquina de estados para el módulo de codificación

La estructura del decodificador son cuatro procesos independientes, en donde se comparan los dos bits mas significativos del bus de datos de recepción y a partir de esto se permite escribir sobre los registros objetivos los cuales son, referencias de voltajes, referencia de corriente, modo de operación y pulsos de disparo de los punte H externos. Luego cada registro es conectado con el módulo correspondiente en donde se interpreta el valor del bus.

Las señales de **entrada** son:

- CLK: Reloj principal de la FPGA.
- RESET: Reset general.
- MSG_TRUE: Bandera que indica que se ha recibido un mensaje en el canal RX correctamente.
- RX_WORD [7:0]: Palabra recibida en el canal RX.

Las señales de **salida** son:

- VC1_REF [1:0]: Referencia del voltaje del condensador C_1.
- VC2_REF [1:0]: Referencia del voltaje del condensador C_2.
- I_REF [11:0]: Referencia para la corriente I_p.
- M_OP [1:0]: Modo de operación.
- SM_1 [1:0]: Disparos para el puente $SM1$.
- SM_2 [1:0]: Disparos para el puente $SM2$.
- EN_SM1: Habilita los disparos para el puente $SM1$.
- EN_SM2: Habilita los disparos para el puente $SM2$.

Las señales **internas** son:

- VC1_REF_B [1:0]: Señal buffer que actualiza el valor de VC1_REF.
- VC2_REF_B [1:0]: Señal buffer que actualiza el valor de VC2_REF.
- M_OP_B [1:0]: Señal buffer que actualiza el valor de M_OP.
- SM_1_B [1:0]: Señal buffer que actualiza el valor de SM_1.
- SM_2_B [1:0]: Señal buffer que actualiza el valor de SM_2.
- I_REF_PT1 [5:0]: Señal que representa los seis bits menos significativos de I_REF.
- I_REF_PT1_B [5:0]: Señal buffer que que actualiza el valor de V_C1_PT1.
- I_REF_B [5:0]: Señal buffer que que actualiza el valor de I_REF.

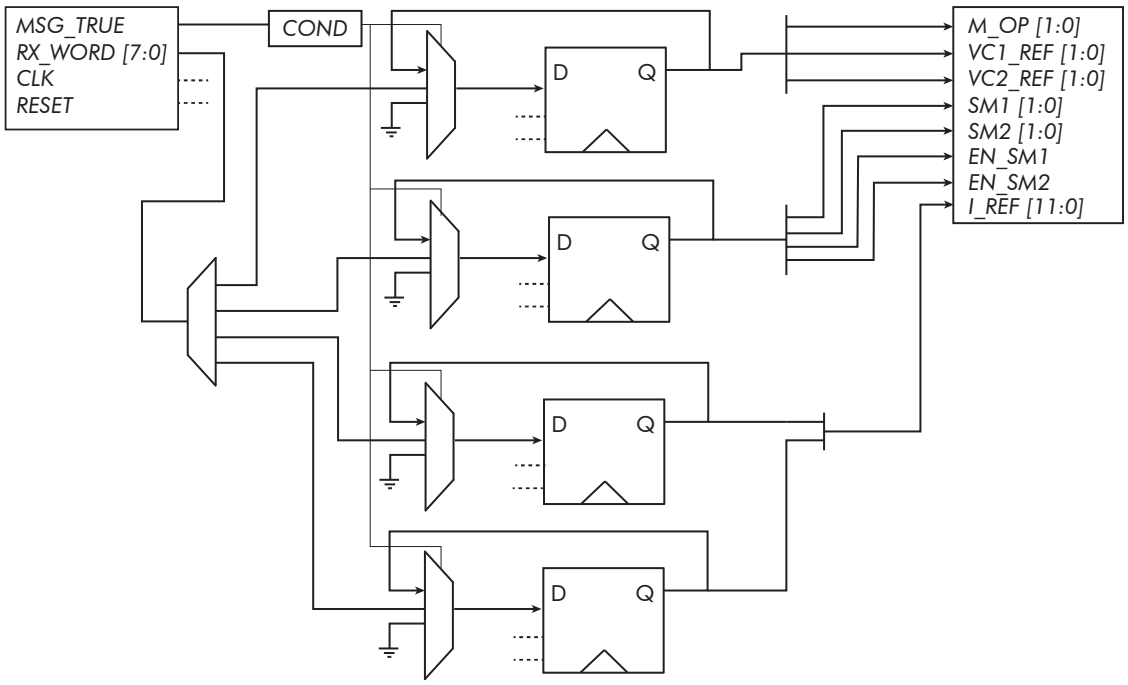


Figura 4.17: Lista de Codificación de Mensajes

CODIFICACIÓN DE MENSAJES A TRANSMITIR

7	6	5	4	3	2	1	0	
0	0	V_C1(5)	V_C1(4)	V_C1(3)	V_C1(2)	V_C1(1)	V_C1(0)	MSG1

7	6	5	4	3	2	1	0	
0	1	V_C1(11)	V_C1(10)	V_C1(9)	V_C1(8)	V_C1(7)	V_C1(6)	MSG2

7	6	5	4	3	2	1	0	
1	0	V_C2(5)	V_C2(4)	V_C2(3)	V_C2(2)	V_C2(1)	V_C2(0)	MSG3

7	6	5	4	3	2	1	0	
1	1	V_C2(11)	V_C2(10)	V_C2(9)	V_C2(8)	V_C2(7)	V_C2(6)	MSG4

CODIFICACIÓN DE MENSAJES A RECIBIR

7	6	5	4	3	2	1	0	
0	0	M_op(1)	M_op(0)	VC2_ref(1)	VC2_ref(0)	VC1_ref(1)	VC1_ref(0)	MSG1

7	6	5	4	3	2	1	0	
0	1	EN_SM1	EN_SM2	SM2(1)	SM2(0)	SM1(1)	SM1(0)	MSG2

7	6	5	4	3	2	1	0	
1	0	I_REF(5)	I_REF(4)	I_REF(3)	I_REF(2)	I_REF(1)	I_REF(0)	MSG3

7	6	5	4	3	2	1	0	
1	1	I_REF(11)	I_REF(10)	I_REF(9)	I_REF(8)	I_REF(7)	I_REF(6)	MSG4

Figura 4.18: Diagrama del decodificador

4.4.4. Lazo de control

En este módulo se implementa el lazo de control encargado del DAB. El módulo recibe las mediciones que son almacenadas en la Interfaz del ADC, las referencias provenientes del decodificador, además de verificar el modo de operación. Las salidas son el valor del ciclo de trabajo y fase correspondientes a los pulsos de disparos que se dirigen a los puentes SP y SS.

Las señales de **entrada** son:

- VC1_REF [1:0]: Referencia de Voltaje del condensador C_1.
- VC2_REF [1:0]: Referencia de Voltaje del condensador C_2.
- M_OP [1:0]: Modo de operación.
- V_C1 [15:0]: Medición de voltaje correspondiente a V_{C1}.
- V_C2 [15:0]: Medición de voltaje correspondiente a V_{C2}.
- IP [15:0]: Medición de voltaje correspondiente a I_p

Las señales de **salida** son:

- DUTY_PWM_SP [31:0]: Referencia del ciclo de trabajo para el modulador PWM del puente SP.
- DUTY_PWM_SS [31:0]: Referencia del ciclo de trabajo para el modulador PWM del puente SS.
- PHASE_PWM_SP [31:0]: Referencia de fase para el modulador PWM del puente SS.

4.4.5. Generador de disparos

Este módulo se encarga de generar los disparos que se dirigen al hardware de potencia. Dentro de este se encuentran cuatro procesos, el primero se encarga de recibir los pulsos de disparos de los puentes H externos, interpretando las señales SM1 y SM2 que vienen desde el decodificador. Con esto se generan los cuatro pulsos correspondientes a cada puente H.
El segundo proceso se encarga de generar los pulsos para el puente H SP, para esto recibe el ciclo de trabajo proveniente del controlador. El proceso genera una señal diente de sierra que es comparada con el valor del ciclo de trabajo para producir los pulsos modulados. Por otro lado, se realiza una comparación entre la fase del modulador del puente H SS y la señal diente de sierra para levantar la bandera que sincroniza el segundo y tercer proceso. En el tercer proceso se generan los pulsos para el puente H SS, de forma similar al proceso descrito anteriormente, se recibe la referencia de ciclo de trabajo proveniente del controlador y la bandera de sincronización para asignar la fase de los pulsos correctamente. En el cuarto proceso se controlan las señales de 'enable' SP_EN, SS_EN que habilitan los 'Gate Drivers' de los puentes SP y SS, a través de la bandera OP_OK que indica que se ha ejecutado correctamente el proceso de inicialización.

Las señales de **entrada** son:

- CLK: Reloj principal de la FPGA.
- RESET: Reset general.
- OP_OK: Señal que indica que se ha terminado el proceso de inicialización.
- EN_SM1: Habilita los disparos para el puente $SM1$.
- EN_SM2: Habilita los disparos para el puente $SM2$.
- SM1 [1:0]: Pulsos de disparos decodificados correspondientes al puente SM1.
- SM2 [1:0]: Pulsos de disparos decodificados correspondientes al puente SM2.

- DUTY_PWM_SP [15:0]: Referencia del ciclo de trabajo para el modulador PWM del puente SP.
- DUTY_PWM_SS [15:0]: Referencia del ciclo de trabajo para el modulador PWM del puente SS.
- PHASE_PWM_SS [15:0]: Referencia de fase para el modulador PWM del puente SS.

Las señales de **salida** son:

- SM1_1: Señal de disparo que se envía al semiconductor SM1_1.
- SM1_2: Señal de disparo que se envía al semiconductor SM1_2.
- SM1_3: Señal de disparo que se envía al semiconductor SM1_3.
- SM1_4: Señal de disparo que se envía al semiconductor SM1_4.
- SM2_1: Señal de disparo que se envía al semiconductor SM2_1.
- SM2_2: Señal de disparo que se envía al semiconductor SM2_2.
- SM2_3: Señal de disparo que se envía al semiconductor SM2_3.
- SM2_4: Señal de disparo que se envía al semiconductor SM2_4.
- SP_EN: Señal de Enable para el Gate Driver del puente SP.
- SS_EN: Señal de Enable para el Gate Driver del puente SS.
- SP_1: Señal de disparo que se envía al semiconductor SP_1.
- SP_3: Señal de disparo que se envía al semiconductor SP_3.
- SS_1: Señal de disparo que se envía al semiconductor SS_1.
- SS_3: Señal de disparo que se envía al semiconductor SS_3.

Las señales **internas** son:

- CONTADOR_PWM [15:0]: Señal que genera las señales de diente sierra para los moduladores PWM.
- PWM_SYNC: Bandera de sincronización de los moduladores PWM para el cálculo de fase del modulador del puente SS.

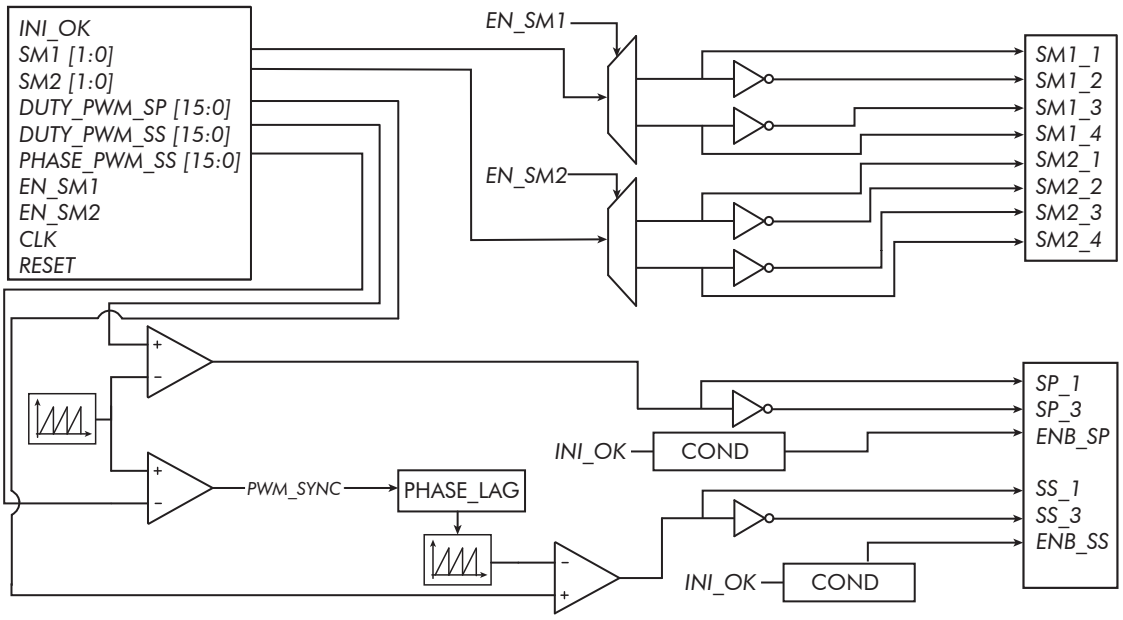


Figura 4.19: Diagrama del Disparador

4.4.6. Interfaz ADC

Este módulo se encarga de comunicar la FPGA con el ADC externo para así almacenar los datos provenientes de los sensores. Para esto se utilizan cuatro procesos, el primero se encarga de generar la señal CNV, encargada de dar inicio al proceso de captura de datos. Luego de pasado el tiempo de adquisición, el proceso levanta la bandera CNV_UP, encargada de habilitar el reloj de conversión y transmisión de datos, el cual es generado por el segundo proceso. El tercer proceso detecta los cantos de subida y bajada del reloj de transmisión proveniente del ADC los cuales levantan la bandera de lectura del canal de datos SD1, SD2 y SD3. En el cuarto proceso se almacenan los valores de los canales en registros que son actualizados en cada lectura y enviados posteriormente al módulo que contiene el lazo de control y codificador según corresponda.

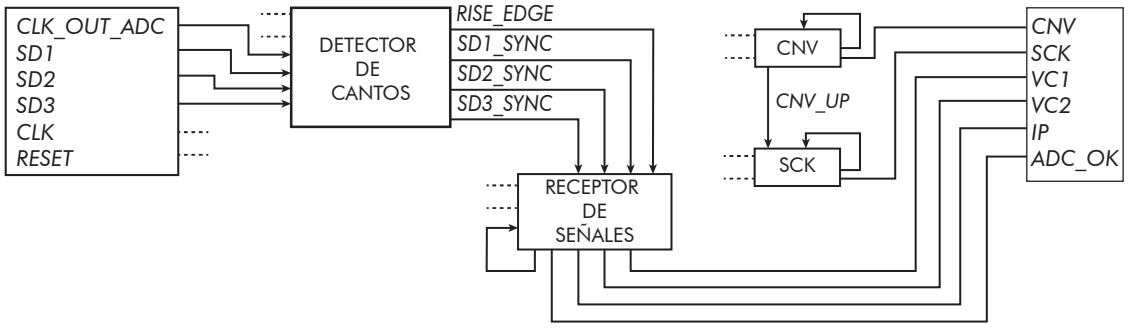


Figura 4.20: Diagrama Interfaz ADC

4.5. Elección de componentes

Para llevar a cabo la implementación es necesario escoger la FPGA y el ADC que puedan cumplir con la síntesis de la implementación y los requisitos expuestos. La FPGA escogida corresponde al modelo MACHXO2-2000HC, este representa un módelo con mayor cantidad de recursos lógicos respescto al modelo elegido en el capítulo 3. El ADC escogido es el modelo LTC2325-12, el cual cuenta con cuatro canales independientes que pueden llegar a una tasa de muestreo de hasta 5 Msps, cumpliendo así los requisitos de diseño.

4.6. Aritmética y simbolización de las señales

Para desarrollar las operaciones matemáticas es necesario sintetizar una una unidad aritmética. El compilador a través de librerías puede interpretar las operaciones básicas, pero la necesidad de utilizar números reales en los cálculos del Lazo de Control conlleva a utilizar un sistema de representación binaria para estos números. Dado que la FPGA escogida no tiene bloques multiplicadores la mejor opción es utilizar una representación de punto fijo para disminuir el número de instrucciones por operación.

La representación escogida es Q6.10, lo que representa 1 bit para el signo, 5 bits para la parte entera y 10 bits para la parte decimal. A continuación se muestran dos ejemplos de la representación:

$$-23{,}5_{10} = 101001{,}0111111111_2 \tag{4.1}$$

$$2{,}625_{10} = 000010{,}1010000000_2 \tag{4.2}$$

El módulo de multiplicación utilizado es generado por la herramienta de síntesis IPexpress que ofrece Lattice Diamond, otras implementaciones basados en el algoritmo de Booth [31] o de Suma y Desplzamiento [32] son recomendadas si es que el sintetizador no es capaz de interpretar la descripción.

4.6.1. Conversión de mediciones

Las señales de voltajes V_{C1} y V_{C2} tienen un rango de 0-600 V las cuales mediante amplificadores operacionales y arreglos resistivos son llevados a una escala de de 0-3.3 V, la que luego es cuantizada por el ADC. Dado que el ADC es de 12 bits la función de transformación correspondiente al voltaje cuantizado es:

$$V_{C_{1,2}} = 0{,}14652 \cdot V_q \tag{4.3}$$

La señal de corriente I_p es medida mediante un sensor de efecto Hall que luego es acondicionada mediante amplificadores operaciones, que transforman el rango de medición de 0-20 A a 0.5-3.3 V. Considerando la cuantización del ADC la función de transformación es:

$$I_p = 0{,}005 \cdot I_q - 0{,}51089 \tag{4.4}$$

Dado que la implementación utiliza la representación Q, los factores de las formulas 4.3 y 4.4 deben representarse utilizando el mismo formato, en consecuencia las fórmulas son reescritas como:

$$V_{C_{1,2}}* = (2^{-3} + 2^{-6} + 2^{-8} + 2^{-9}) \cdot V_q \tag{4.5}$$

$$I_p* = (2^{-8} + 2^{-10}) \cdot I_q - 2^{-1} \tag{4.6}$$

Los valores de voltajes y corrientes que son usados en el Lazo de Control son referidos por unidad, por conveniencia se eligen valores base que son potencias de dos dada su simplicidad para la conversión.

Por lo que la conversión queda definida como:

$$V_{pu} = \frac{V_{C_{1,2}}*}{512V} \tag{4.7}$$

$$i_{pu} = \frac{I_p*}{16A} \tag{4.8}$$

5 Diseño electrónico de la Tarjeta de Control

En este capítulo se realiza el diseño PCB de la tarjeta de control. Este comienza con la descripción de los elementos que constituyen la placa, la interconexión de los elementos es detallada mediante los diagramas de buses. Finalmente se presentan los esquemáticos y las dimensiones de la placa.

5.1. Características

Esta placa electrónica esta pensada para implementar el sistema de control descrito en el Capítulo 4. La placa contiene la FPGA MACHXO2-2000HC y el ADC LTC2325-12. La FPGA cuenta con una memoria Flash no volátil integrada y una SRAM para el almacenamiento de la descripción de hardware. Además contiene un oscilador interno con una frecuencia máxima de hasta 133 MHz y un oscilador externo de 125 MHz para tareas de precisión. Para la programación de esta, se debe conectar el programador HW-USBN-2B al puerto JTAG. El ADC de 12 bits y de cuatro canales independientes es controlado por la FPGA, con una velocidad de muestreo de 2 Msps ajustable hasta una velocidad máxima de 5 Msps. La placa cuenta con un puerto de comunicación UART full dúplex mediante fibra óptica.

La placa de control incluye:

- La FPGA LCMXO2-2000HC-6TG100C. El ADC externo de 12 bits LTC2325-12.
- Puerto 8 pin header JTAG para la programación de la FPGA.
- Puerto 2x15 pin I/O para integración con hardware de potencia.
- Jack USB mini B para alimentación 5V.

5.2. Diagrama de alimentación

La tarjeta puede ser alimentada desde el puerto USB con un voltaje de 5 V, el convertidor DC/DC PDS2-S5 reduce la tensión a 3.3 V para así alimentar la electrónica de la tarjeta. Por otro lado, la tarjeta puede ser alimentada desde el pin 26 del header J2, con un voltaje de 3.3 V. Se recomienda alimentar la tarjeta desde el puerto USB durante la programación y usar alimentación mediante el header J2 cuando esta es conectada al hardware de potencia mediante el cable flex plano.

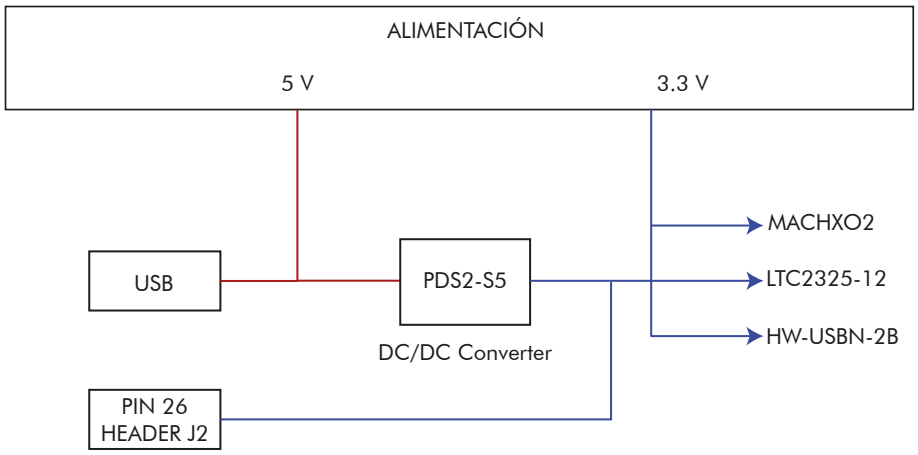


Figura 5.1: Diagrama de alimentación Placa de Control

5.3. Diagrama de buses

El diagrama de la Figura 5.3 muestra las rutas de los principales buses que interconectan los elementos de la placa. En las tablas 5.1 y 5.2 se muestran las asignaciones para el puerto de programación y el puerto de interconexión con el hardware de potencia correspondientemente.

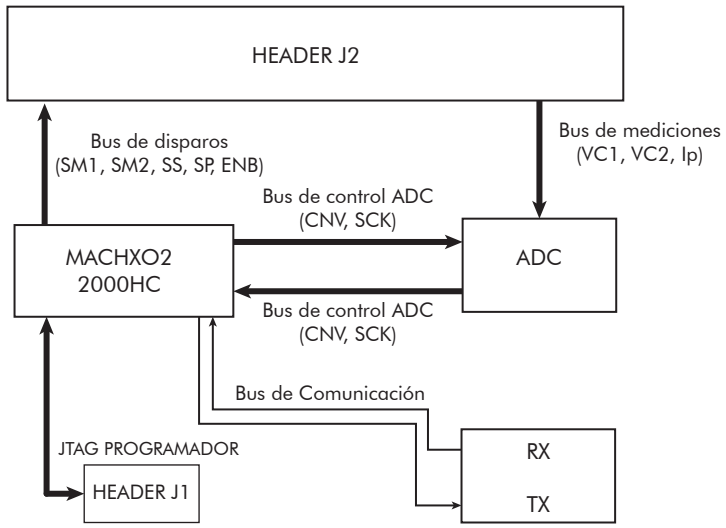


Figura 5.2: Diagrama de buses Placa de Control

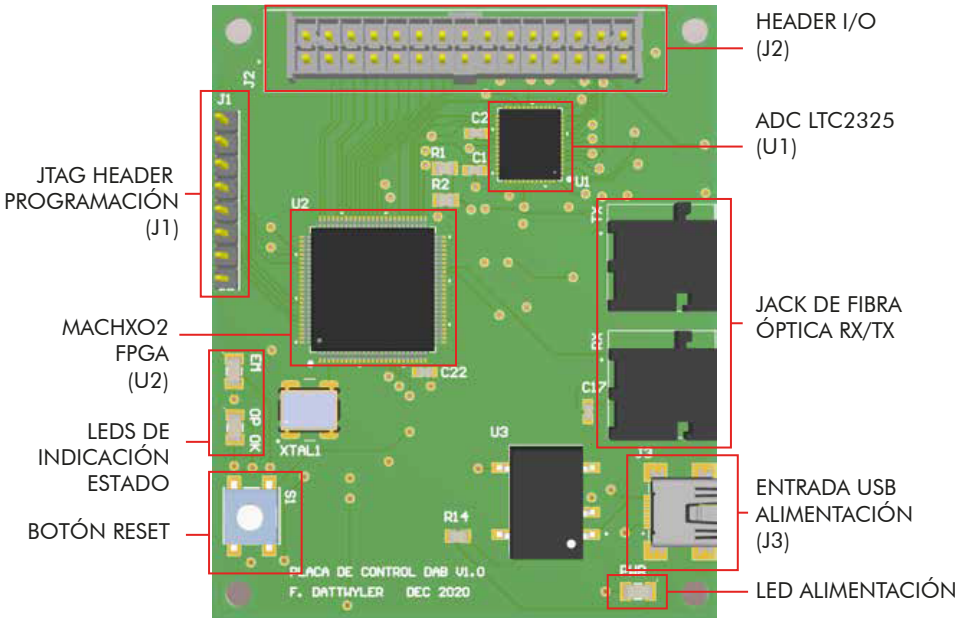


Figura 5.3: Vista de superficial Tarjeta de control

Tabla 5.1: Pinout Header J1

n° PIN	**Función**
1	VDD
2	TDO
3	TDI
4	TCK
5	TMS
6	JTAGEN
7	GND
8	NC

Tabla 5.2: Pinout Header J2

n° PIN	**Función**	**n° PIN**	**Función**
1	NC	16	NC
2	AN3+ (I_p)	17	NC
3	SM1_1	18	NC
4	SM1_2	19	ENB_SP
5	SM1_3	20	ENB_SS
6	SM1_4	21	NC
7	SP_1/2	22	NC
8	SP_3/4	23	AN1+ (V_{C1})
9	SS_1/2	24	AN2+ (V_{C2})
10	SS_3/4	25	GND
11	SM2_1	26	VCC
12	SM2_2	27	NC
13	SM2_3	28	NC
14	SM2_4	29	NC
15	NC	30	NC

5.4. Esquemáticos

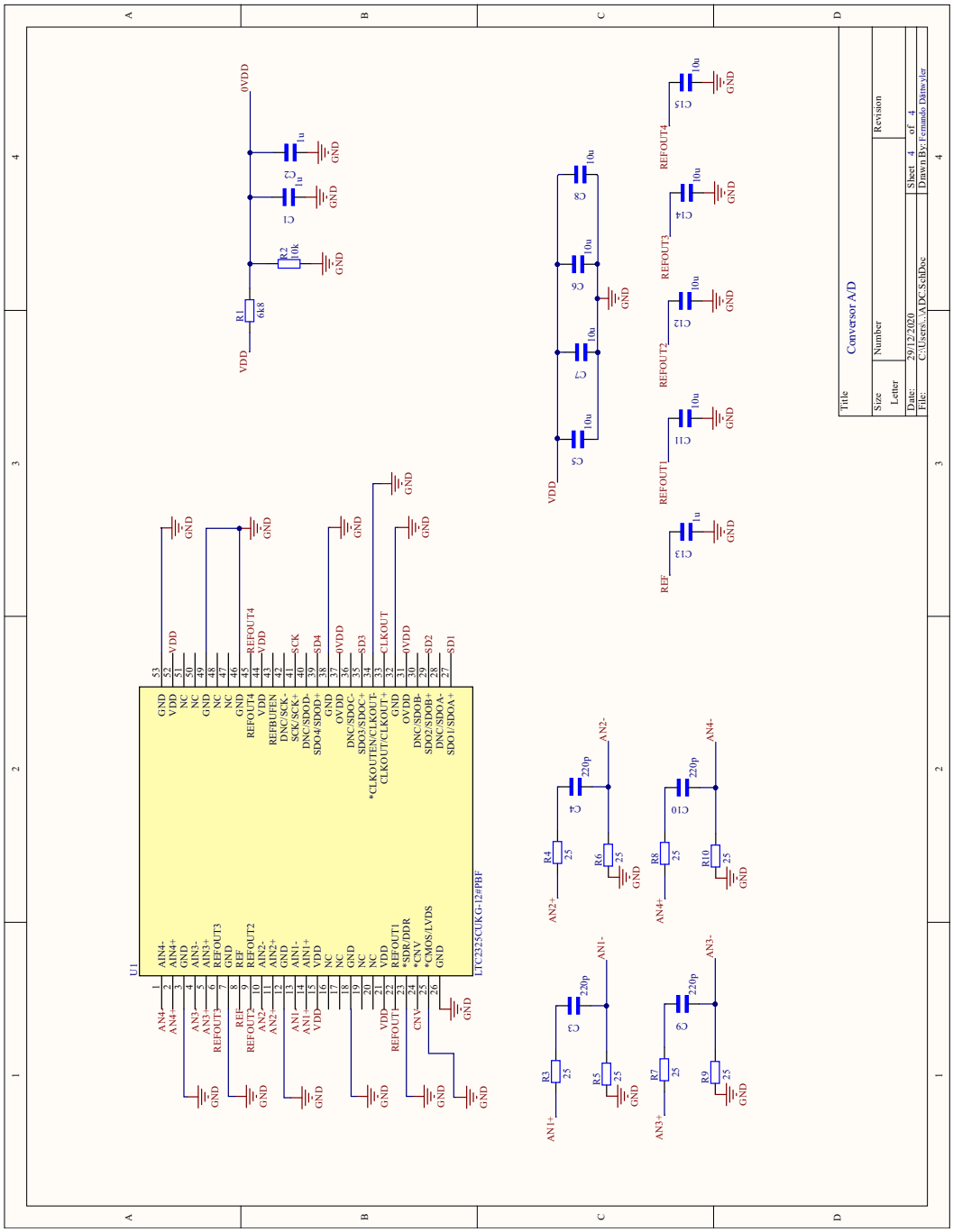


Figura 5.4: Esquemático del conversor análogo digital

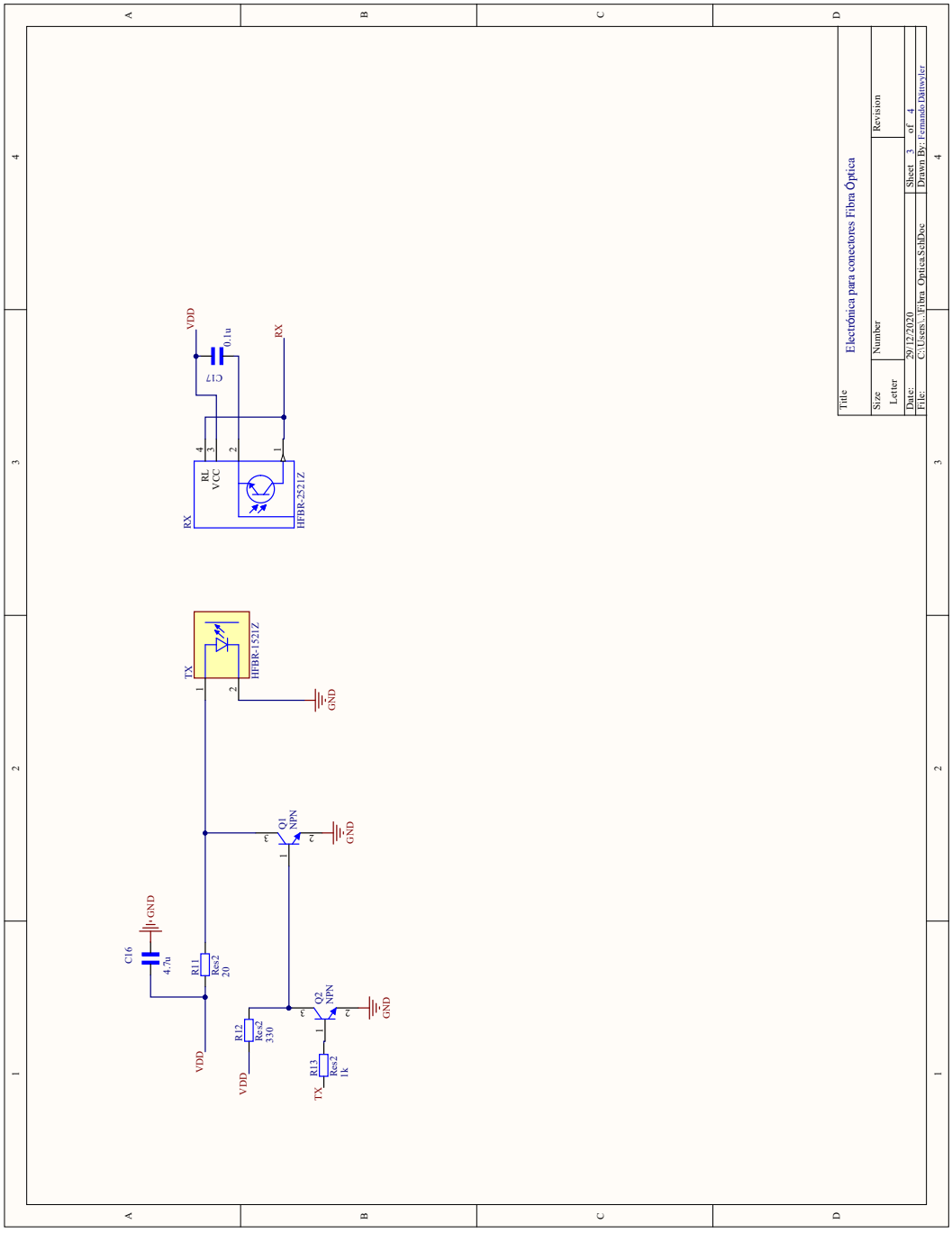


Figura 5.5: Esquemático de los conectores de fibra óptica

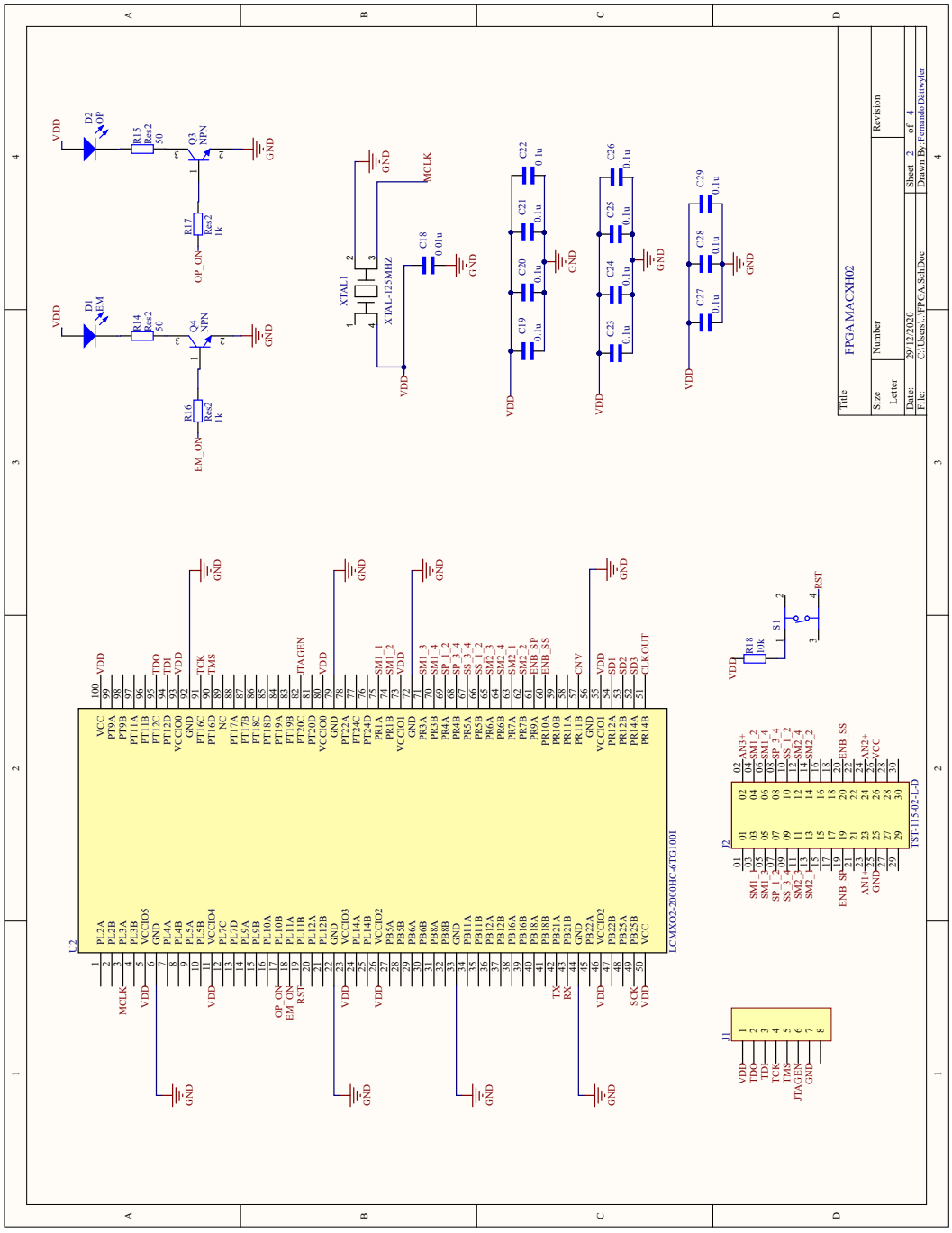


Figura 5.6: Esquemático de la FPGA MACHXO2-2000

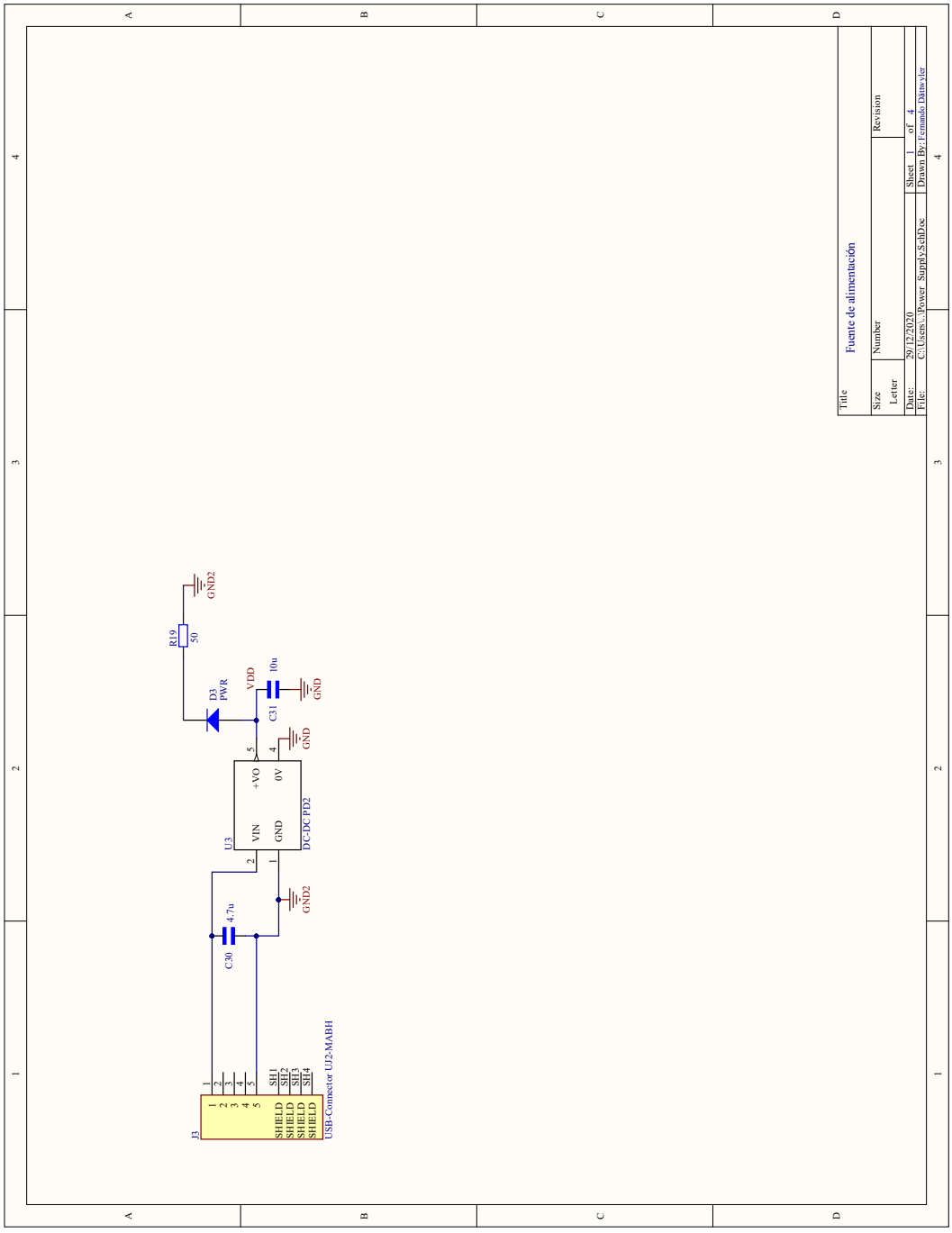


Figura 5.7: Esquemático de la fuente de alimentación

5.5. Lista de Materiales

En la Tabla 5.3 se muestra la lista de materiales para la Tarjeta de Pruebas

Tabla 5.3: Lista de materiales Tarjeta de Control

Item #	**Etiqueta**	**N° de pieza del fabricante**	**Cantidad**
1	C1, C2, C13	C0603C105K9PACTU	3
2	C3, C4, C9, C10	C0603C221K3RACTU	4
3	C5, C6, C7, C8, C11, C12, C14, C15, C31	C0603C106M7PAC7867	9
4	C16, C30	C0603C475K8PACTU	2
5	C17, C19, C20, C21, C22, C23, C24, C25, C26, C27, C28, C29	C0603C104K8RACTU	12
6	C18	C0603C103K5RACTU	1
7	D1,D2,D3	BL-HB333Q-AV-TRB	3
8	J1	68002-408HLF	1
9	J2	IPL1-115-01-L-D-K	1
10	J3	UJ2-MBH-1-SMT-TR	1
11	Q1, Q2, Q3, Q4	2SCR553RTL	4
12	R1	RC0805JR-076K8L	1
13	R2, R18	RC0805JR-0710KL	2
14	R3, R4, R5, R6, R7, R8, R9, R10	ERJ-P06F24R9V	8
15	R11	ERJ-P06J200V	1
16	R12	RC0805JR-07330RL	1
17	R13, R16, R17	RT0805BRD071KL	3
18	R14, R15, R19	RT0805BRD0749R9L	3
19	RX	HFBR-2521Z	1
20	S1	430451025836	1
21	TX	HFBR-1521Z	1
22	U1	LTC2325IUKG-12#PBF	1
23	U2	LCMXO2-2000HC-6TG100I	1
24	U3	PDS2-S5-S3-M-TR	1
25	XTAL1	ASV-125.000MHZ-EC-T	1

5.6. Dimensiones y Vistas

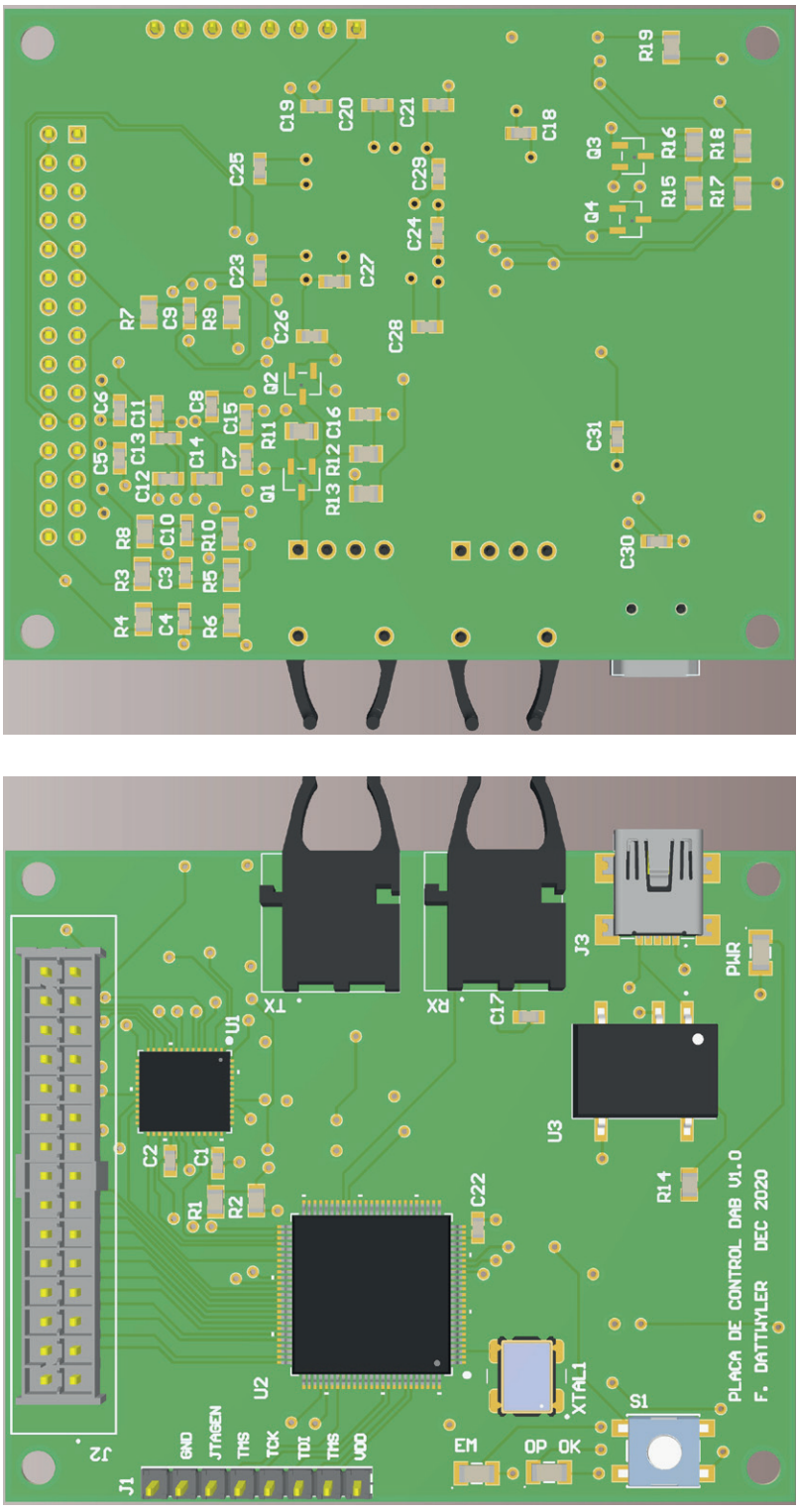


Figura 5.8: Vistas superior e inferior de la Tarjeta de Control

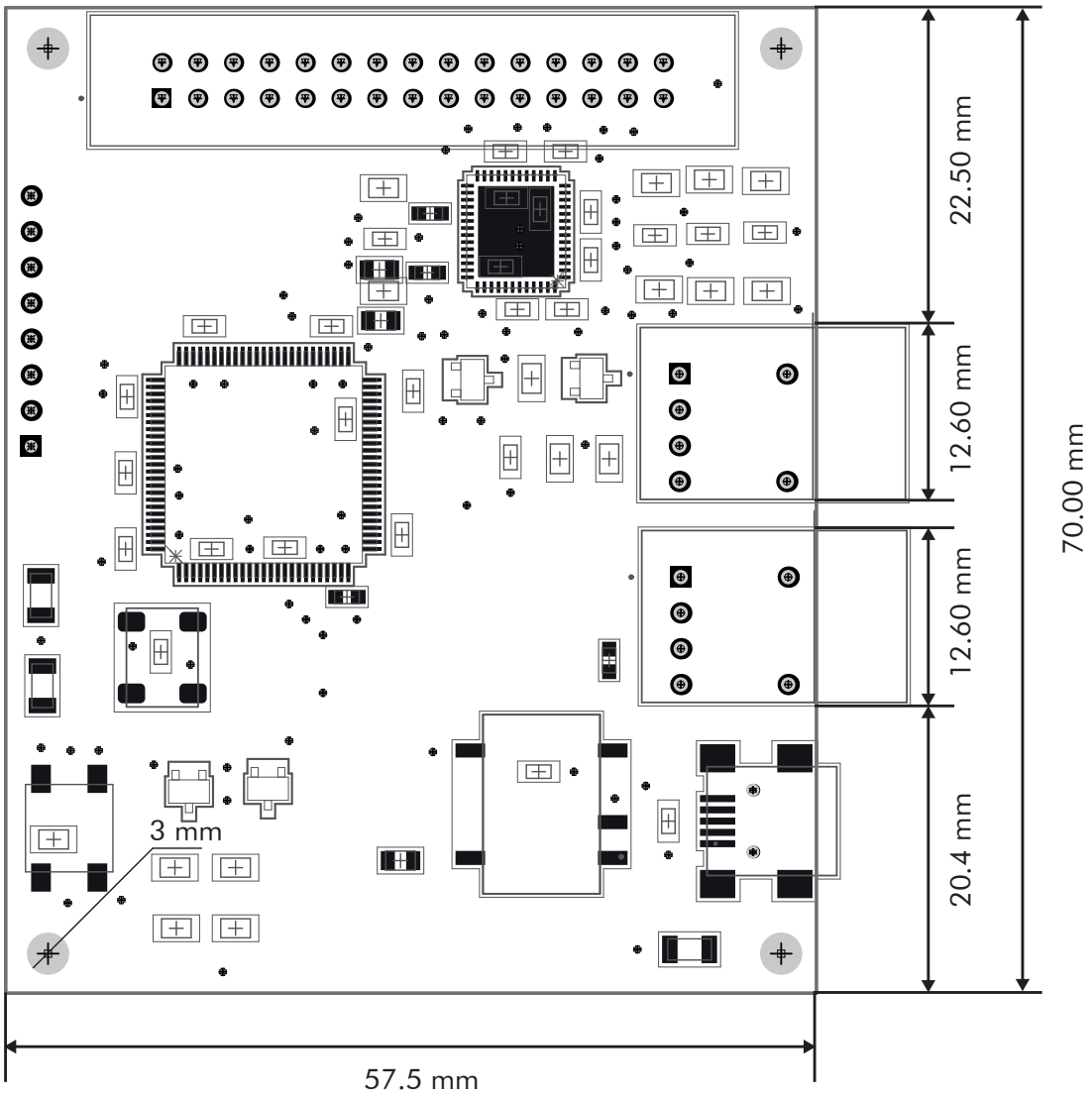


Figura 5.9: Dimensiones físicas de la Tarjeta de Control

6 Resultados

En el Capítulo 4 fue presentado un sistema modular capaz de satisfacer los requisitos de funcionamiento para la tarjeta de control. En este capítulo se presentarán los resultados de la síntesis de los módulos propuestos en la implementación. Para ello se mostrarán las simulaciones y mediciones de las principales señales obtenidas y resultados de síntesis. Finalmente se presenta la cotización de la orden de fabricación para las tarjetas de pruebas y control.

6.1. Herramientas de simulación y medición

Para las simulaciones se ha usado el software Aldec-Active-HDL y los códigos han sido implementados en la placa de evaluación LCMXO2-7000HE-B-EVN. Las señales son medidas utilizando el analizador lógico SALEAE 8 CH/24 MHz, posteriormente los datos son exportados en formato CSV para ser graficados en el software MATLAB. El reloj del simulador y el oscilador interno de la placa de evaluación funcionan a una velocidad 133 MHz para la evaluación de los módulos.

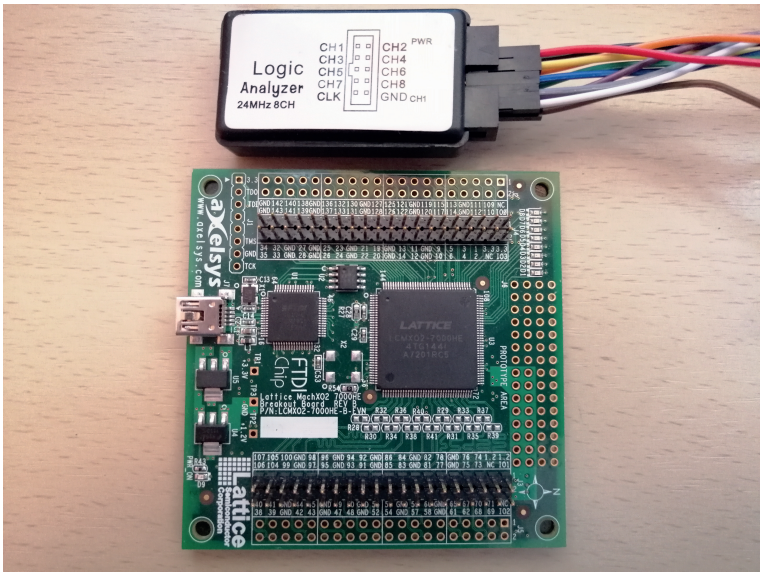


Figura 6.1: Placa de evaluación y analizador lógico

6.2. Pruebas de comunicación Módulo UART

Para comprobar el funcionamiento del módulo UART se realiza una prueba de Eco, lo que significa que el canal receptor leerá lo que se está transmitiendo. Se transmitirán los mensajes 0x01, 0x42 y 0xC4, con un baudrate de transmisión de 10 Mhz. En las Figura 6.2 y 6.3 se muestra el resultado de la simulación y en la Figura 6.4 la medición del canal de recepción, la bandera de lectura del canal y la bandera de confirmación de la recepción exitosa. en la Tabla 6.1 se muestran las mediciones referentes al baudrate y el tiempo mínimo para poder enviar un nuevo mensaje.

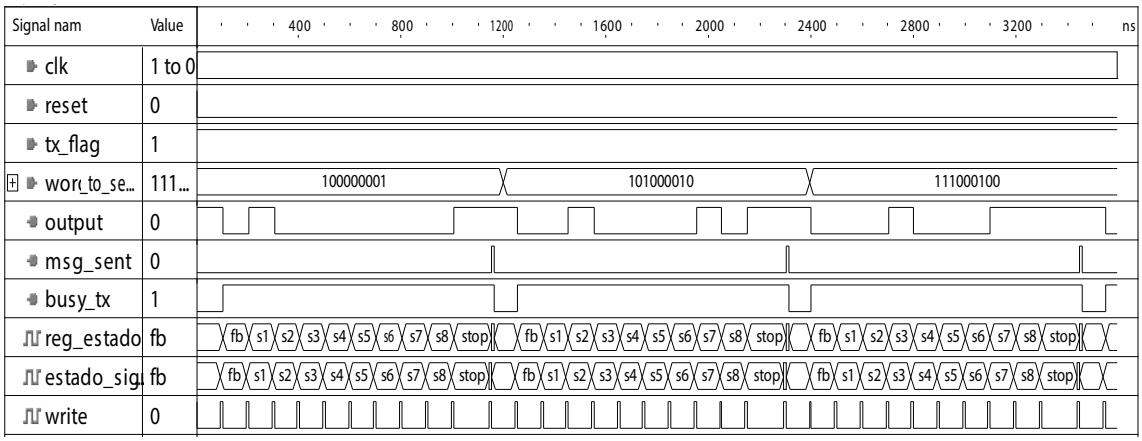


Figura 6.2: Simulación Modulo UART TX

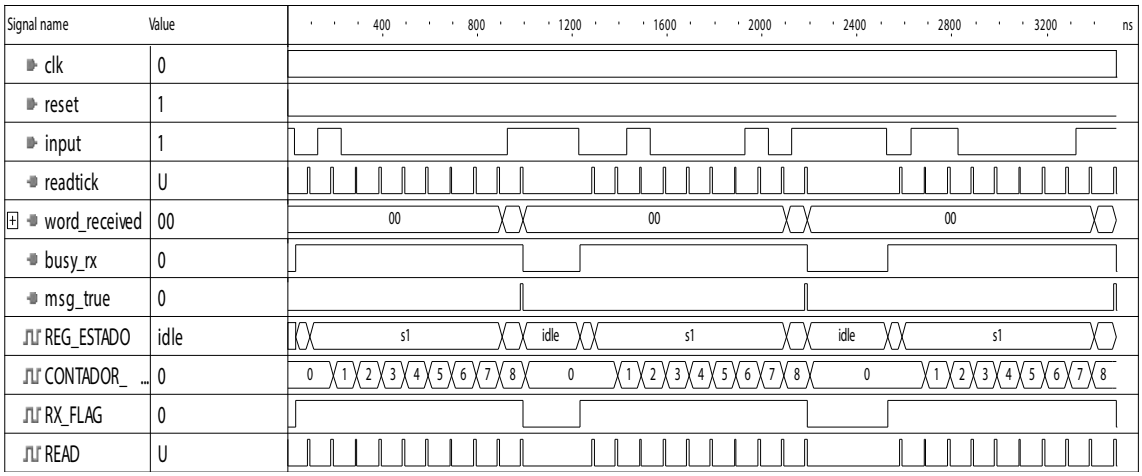


Figura 6.3: Simulación Modulo UART RX

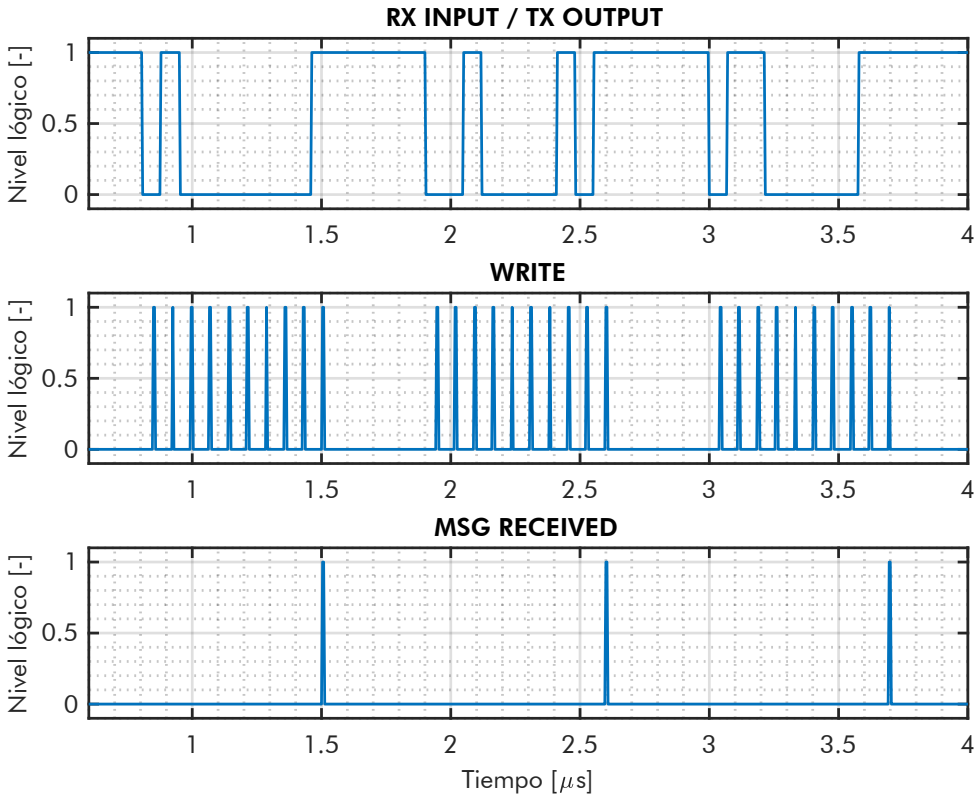


Figura 6.4: Medición del módulo UART

Tabla 6.1: Mediciones de pruebas módulo UART

Parámetro	Medición	Unidad
Baudrate Deseado	10	MHz
Baudrate Promedio Medido	11.001	MHz
Tiempo mínimo entre mensajes	75.20	ns

6.3. Pruebas del Módulo de Disparos

La prueba del módulo de disparos consiste en generar mensajes externos para el encendido y apagado que son interpretados por el Decodificador y comprobar el funcionamiento para ciclos de trabajos establecidos. Para los disparos del puente SM1 se fija un ciclo de trabajo del 8.3 % y para el puente SM2 se fija un ciclo de trabajo del 50 %. En la Figura 6.5 se muestran los resultados medidos a través del analizador lógico, los valores asociados a la frecuencia y el ancho de pulso de los disparos se muestran en la Tabla 6.2.

Para comprobar las señales de disparos que se dirigidas al DAB, se fijan los valores de referencia en el Lazo de control. El valor de prueba para el ciclo de trabajo del puente SP es de 50 %, el ciclo de trabajo correspondiente al puente SS es de de 16.6 % y la fase de $\frac{\pi}{4}$. Con esto se prueba el ajuste del ciclo de trabajo, independencia de funcionamiento y fase ajustable para los disparos.

Las mediciones se muestran en la Figura 6.6 , los valores asociados a la frecuencia y el ancho de pulso de los disparos son recopilados en la Tabla 6.3.

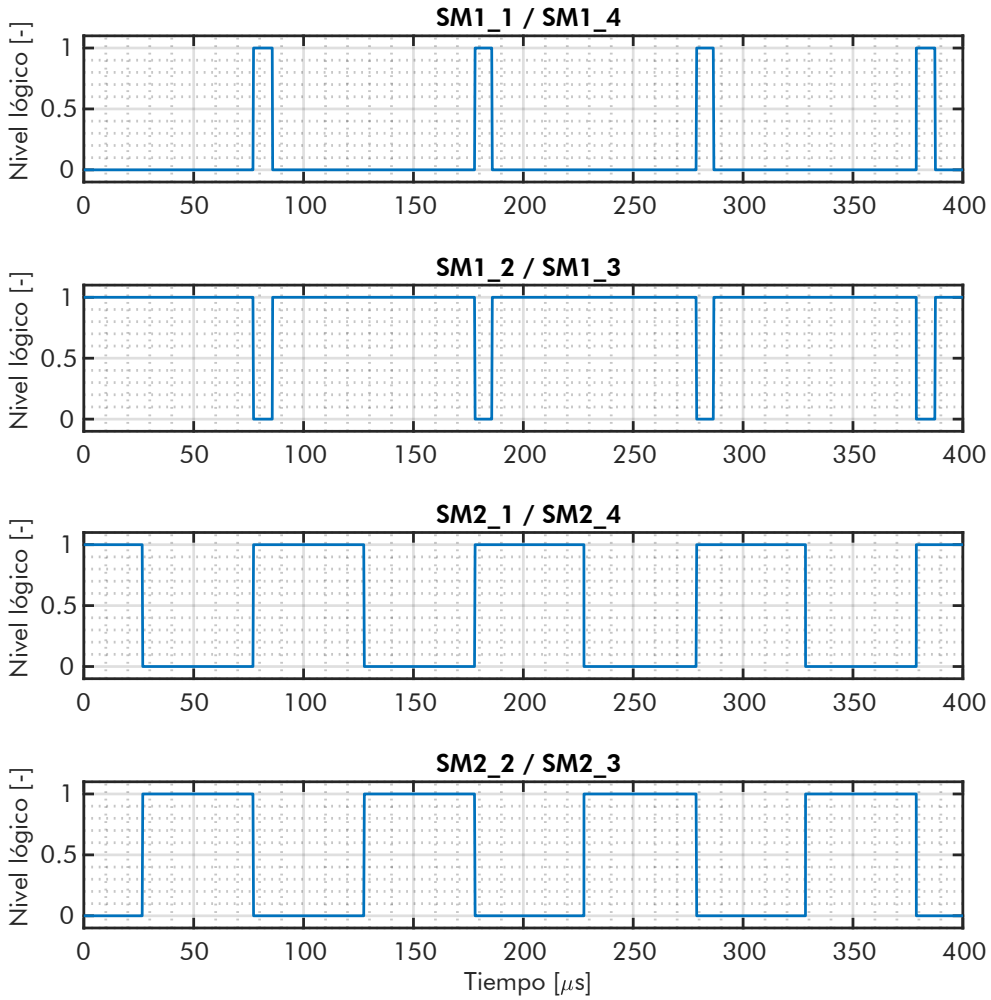


Figura 6.5: Secuencia de pulsos de disparo para los puentes SM1 y SM2

Tabla 6.2: Mediciones de disparos a los puentes SM1 / SM2

Parámetro	Medición	Unidad
Frecuencia de los pulsos de disparo deseada	10	kHz
Frecuencia de los pulsos de disparo obtenida	10.2	kHz
Error en el ancho de pulso	<1 %	-
Ancho de pulso mínimo	2 %	-

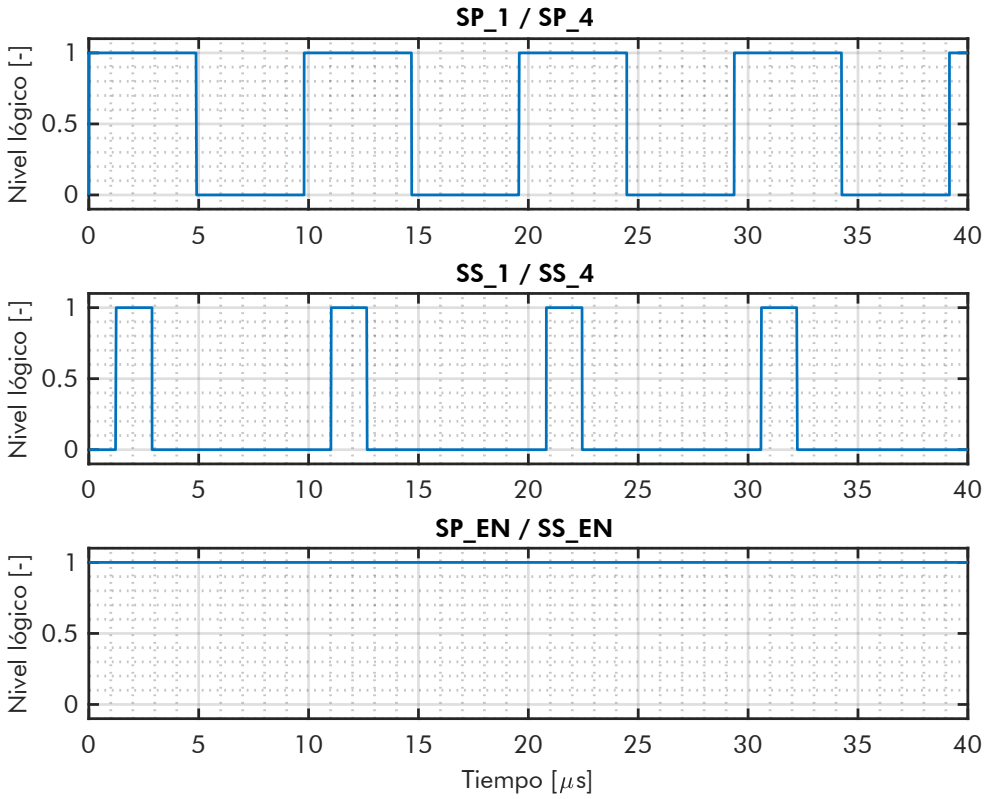


Figura 6.6: Secuencia de pulsos de disparo para los puentes SP y SS

Tabla 6.3: Mediciones de disparos a los puentes SP / SS

Parámetro	Medición	Unidad
Frecuencia de los pulsos de disparo deseada	100	kHz
Frecuencia de los pulsos de disparo obtenida	113	kHz
Error en el ancho de pulso	<0.2 %	-
Ancho de pulso mínimo	0.2 %	-

6.4. Pruebas módulo de Control central

Para comprobar el funcionamiento del módulo de Control Central se realiza la función de inicialización la cual toma 1 ms, luego se levanta una bandera de sobre voltaje en $t = 1{,}20$ ms que transiciona la máquina desde el estado de Operación a Emergencia, se vuelve al estado Idle en $t = 1{,}25$ ms mediante el cambio del modo de operación '11'. Finalmente se genera una bandera de sobre corriente en $t = 1{,}30$ ms dejando a la máquina en el estado de emergencia. El proceso descrito anteriormente es ilustrado en la Figura 6.7.

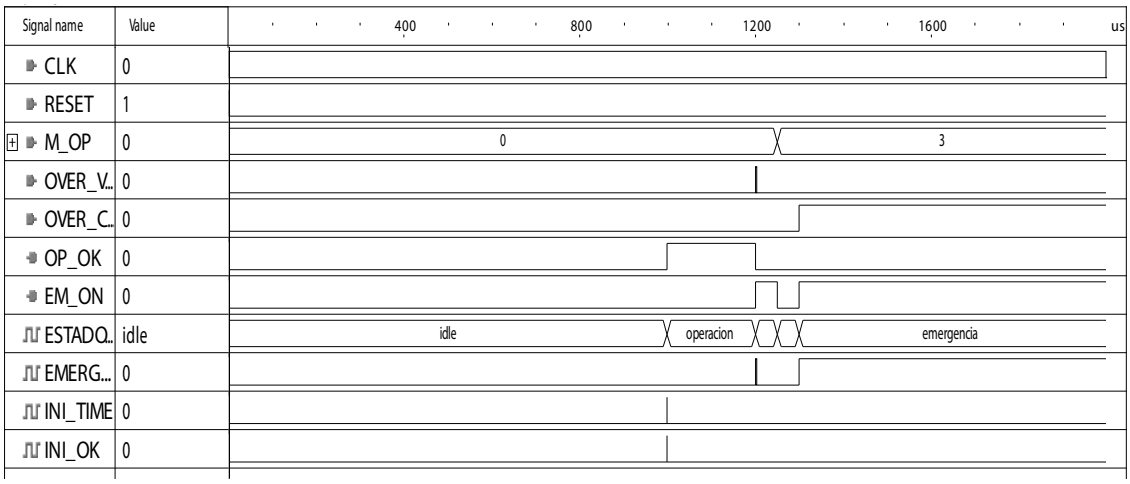


Figura 6.7: Prueba Módulo de control Central

6.5. Pruebas Decodificador

Para probar el módulo de Codificación se generan dos valores de voltaje provenientes de la Interfaz ADC correspondiente a los valores 0xFD y 0xFE para los registros V_C1 y V_C2 en $t = 0$ ns, los que cambian a 0x4FF y 0x4FC en $t = 100$ ns. A partir de la tabla de codificación los valores a asignar en los registros a transmitir en $t = 0$ ns son:

- VC1_pt1= '00111101'
- VC1_pt2= '01000011'
- VC1_pt1= '10111110'
- VC1_pt2= '11000011'

Y los valores a transmitir en $t = 100$ ns son:

- VC1_pt1= '00111111'
- VC1_pt2= '01010011'
- VC1_pt1= '10111100'
- VC1_pt2= '11000011'

La prueba correspondiente al Decodificador consiste en generar mensajes provenientes del receptor y observar si la decodificación de estos se realiza de forma correcta, almacenando los valores de los mensajes en los registros objetivos según la tabla de codificación de la Figura 4.18. En la Figura 6.9, se observa la recepción de cuatro mensajes que llegan cada 50 ns con valores 0x2D, 0x4C, 0x8E y 0xF6. Cada uno corresponde a un tipo de mensaje de la tabla de codificación asignando los siguientes valores:

- VC1_REF= '01'
- VC2_REF= '11'
- M_OP= '01'
- SM1= '00'
- SM2= '11'
- EN_SM1= '0'
- EN_SM2= '0'
- I_REF= '110110001110'

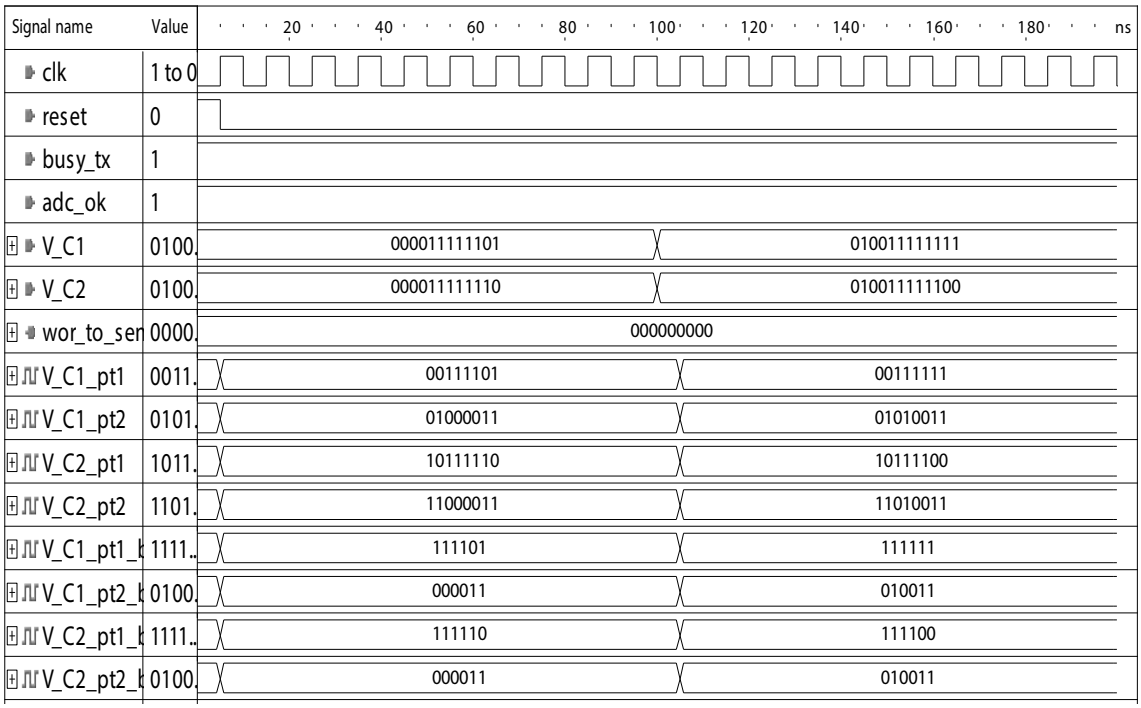


Figura 6.8: Prueba de codificación de mensajes

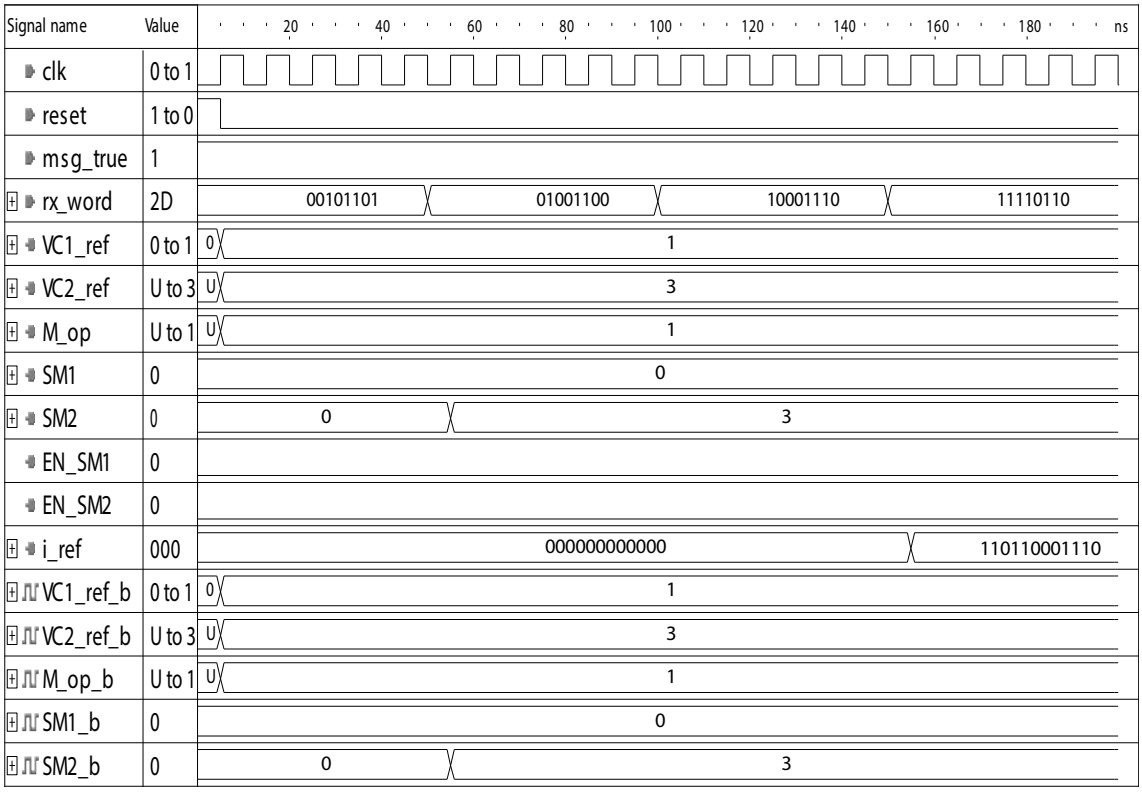


Figura 6.9: Prueba de decodificación de mensajes

6.6. Pruebas de Interfaz ADC

Para comprobar el funcionamiento de este módulo se generarán señales que debe enviar la FPGA al ADC y además se sintetizarán las señales que entregaría este último para comprobar el sistema de almacenamiento de datos en los registros objetivos. Los valores se agregarán como condicional de encendido a la bandera ADC_OK para su corroboración.

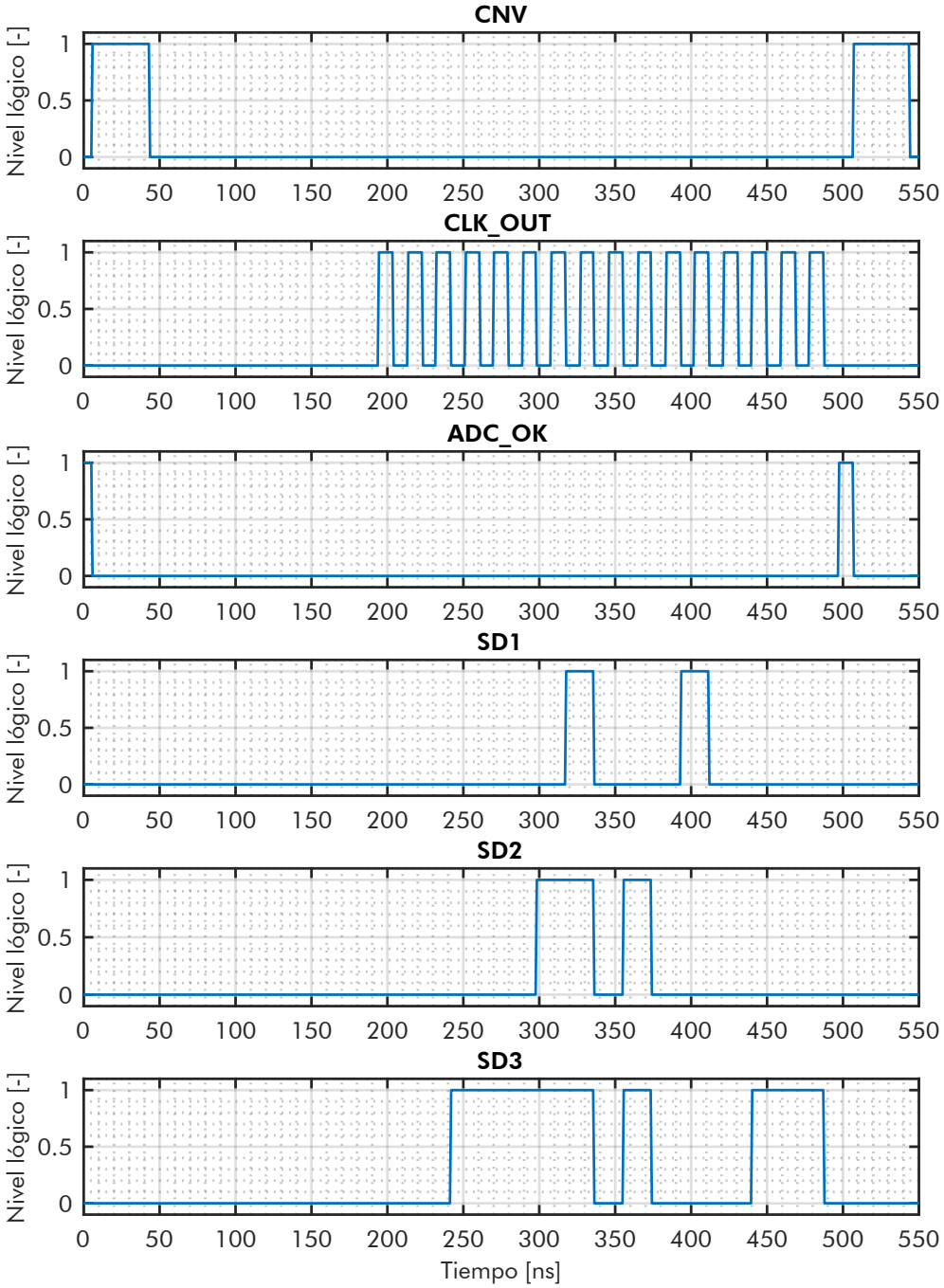


Figura 6.10: Máquina de estados para el módulo de recepción

6.7. Tiempos de Inicialización y Ruta Crítica

En esta sección se mostrarán las mediciones de los tiempos de las tareas inicialización y la ruta crítica que fueron expuestos en el Capítulo 4. En la Tabla 6.4 se muestran los tiempos referidos de la Figura 4.2 correspondientes a la rutina de inicialización. En la Tabla 6.5 se especifican los tiempos de la ruta crítica correspondiente a los procesos de las Figuras 4.3, 4.4, 4.5 y 4.6

Tabla 6.4: Tiempos de las tareas de Inicialización

Parámetro	Descripción	Mín	Tip	Máx	Unidad
T_a	Tiempo total para la recepción de un mensaje.	1.01	-	-	μs
T_b	Tiempo máximo para la transmisión del primer mensaje.	-	-	0.5	μs
T_c	Periodo de transmisión de un mensaje	-	1	-	μs
T_d	Periodo de muestreo del ADC	-	0.5	-	μs
T_e	Tiempo requerido para enviar el primer pulso de disparo	1.01	-	4.06	μs

Tabla 6.5: Tiempos asociados a la ruta crítica del sistema

Parámetro	Descripción	Mín	Tip	Máx	Unidad
T_1	Tiempo necesario para recibir, almacenar, convertir y transmitir las mediciones de V_{C1} y V_{C2} al sistema de control externo.	-	4.54	-	μs
T_2	Tiempo necesario para recibir y decodificar los mensaje que contienen referencias, pulsos de disparos a los puentes externos y modo de operación.	-	4.04	-	μs
T_3	Actualización del estado y envío de las señales de disparo de los puentes externos	-	15.08	-	ns
T_4	Tiempo necesario para obtener nuevas referencias de ciclos de trabajo y fase para los puentes SP y SS	-	10 μs -T_t	-	μs
$T_5^{(1)}$	Tiempo para la recepción y envío de un pulso de disparo dirigido al DAB con diferente ciclo de trabajo y fase.	7.54	-	$1 \cdot 10^4$	ns
T_t	Suma de los tiempos T_1 y T_2	-	8.58		μs

(1) El tiempo correspondiente puede variar desde dos ciclos de reloj si es que el puente se encuentra deshabilitado o esperará hasta el próximo periodo del pulso de disparo.

6.8. Recursos del Diseño Lógico

A continuación se presentan los resultados de los recursos ocupados en la implementación del diseño lógico en la FPGA entregado por la herramienta de síntesis.

Tabla 6.6: Recursos utilizados por la FPGA

Parámetro	Medición	Porcentaje
Uso de Registros	584/2352	25 %
Uso de Slices	677/1056	64 %
Uso de LUTs	1311/2112	62 %

6.9. Costos de Producción

En la Tabla 6.7 se muestran los resultados de la cotización de las componentes y fabricación de las PCBs. El distribuidor de componentes es DIGIKEY y el fabricante consultado es PCBWAY. El costo de componentes corresponde al de una sola placa mientras que el costo de de fabricación corresponde a 10 unidades que representan el costo mínimo de fabricación.

Tabla 6.7: Costos de producción

Parámetro	Medición
Costo componentes Tarjeta de Pruebas	47,22 US
Costo fabricación Tarjeta de Pruebas	141 US
Costo componentes PCB Tarjeta Control	94,15 US
Costo fabricación PCB Tarjeta Control	133 US

7 Conclusiones

En el trabajo presentado en la Memoria, se ha desarrollado una tarjeta para el control de un módulo SST, la cual considera la captura y procesamiento de los datos provenientes desde el hardware de potencia, la comunicación con el sistema de control externo y la generación de disparos para los semiconductores.

El diseño de la tarjeta de pruebas logró el aprendizaje del software ALTIUM para el diseño PCB, obteniendo los esquemáticos, lista de materiales y los archivos Gerber para su orden de fabricación. Sin embargo la programación de los integrados y la evaluación de rendimiento, se tuvo que llevar a cabo con la utilización de tarjetas de evaluación y softwares de simulación, debido a la imposibilidad de ingresar al laboratorio por las condiciones de la pandemia. A partir de estos resultados, se decide que la solución debe ser compuesta por una FPGA y un ADC externo.
Esta combinación de integrados posibilita que las tareas y requisitos puedan ser realizadas en paralelo minimizando el tiempo de la ruta crítica.

Las tareas son organizadas por módulos ordenados de forma jerárquica, la cual es presentada en la Figura 4.8. Para cada módulo se realizan pruebas corroborando su funcionamiento. La Unidad de control central cumple satisfactoriamente con las tareas de inicialización y acciones ante detección de fallas, respecto a la tarea de inicialización se ha habilitado el voltaje de los condensadores como variable en la lógica de transición, considerando una configuración futura del voltaje de operación nominal como condición de paso al modo de OPERACION.

El módulo UART si bien transmite y recibe los mensajes de forma correcta, es sensible ante variaciones del reloj principal, como consecuencia de su precisión y división no exacta mediante los contadores de variable entera, es de importancia obtener un valor preciso de la tasa de transmisión, para configurar el módulo de comunicación del sistema de control externo al no ser una comunicación sincrónica.

La forma de codificación y decodificación mediante símbolos representa una forma eficiente para la transmisión de información. Esta es flexible y permite cambiar el significado e interpretación de cada símbolo de forma sencilla en los módulos correspondientes. Para la transmisión de las mediciones se ha limitado a 12 bits la precisión de los valores, por lo que se transmiten las tramas provenientes directamente del ADC.

El módulo de disparos permite controlar de forma independiente los disparos que se dirigen a cada puente H. La precisión de los disparos a los puentes SM1 y SM2 son dependientes de los mensajes recibidos por el módulo UART, por lo que la frecuencia de los pulsos y el ancho mínimo están determinados por el tiempo necesario para poder recibir un mensaje y el retardo de actualización de los registros. Por otro lado, los disparos que se dirigen a los puentes del DAB son sensibles al reloj principal de forma similar al módulo UART, el ancho de pulso mínimo esta directamente relacionado con el paso del contador que forma la señal diente de sierra del modulador y la frecuencia a la precisión del reloj principal y la división hecha por los contadores de variable entera. El error en las frecuencias y anchos de pulsos son tolerables, aun así, ajustes en los parámetros de los contadores y el uso de un oscilador externo pueden ayudar a aumentar la precisión de los disparos.

Para el módulo de interfaz ADC se han generado las señales que controlan el ADC externo, generado las señales que este último entregaría según las especificaciones de su hoja de datos, por último se ha comprobado que los datos son correctamente recibidos con la bandera ADC_OK.

Los recursos utilizados de la FPGA son mostrados en la Tabla 6.6, en la que se observa que aún se encuentra espacio disponible y suficiente para la síntesis del módulo del lazo de control. Es difícil estimar el consumo de cada módulo ya que la herramienta de síntesis optimiza el uso de estos recursos, modificando los resultados según la interconexión de los módulos, pero la herramienta IPexpress permite ver el número de LUTs y Slices que tomará la implementación de los bloques multiplicadores los cuales representan un consumo intensivo de lógica.

El diseño es finalizado con la generación de esquemáticos, PCB layout y archivos Gerber, considerando las recomendaciones de las hojas de datos para la electrónica y confección de footprints. La placa resultante tiene dimensiones 57.50 x 70.00 mm, contiene cuatro capas de las cuales la superior e inferior son usadas para el ruteo de pistas, las capas internas para tener un plano de alimentación y tierra. La alimentación de esta puede ser de 5V a través del jack USB o de 3.3 a través del puerto de conexión J2. EL costo de las componentes para una unidad es de 47.22 $US y la fabricación de 10 unidades es de 133 $US, estos costos son variables ya que se ofrecen descuentos por volumen tanto para las componentes como para las PCBs.

El proyecto debe considerar las siguientes líneas de trabajo futuro:

- La síntesis del controlador y su implementación en la FPGA respetando las señales de entrada y salida descritas en la Sección 4.4.4 y considerando los tiempos de la ruta crítica expuestos en la Tabla 6.5.
- Luego del proceso de ensamblaje de componentes en la placa, es necesario volver a realizar las pruebas de cada módulo y observar las diferencias de funcionamiento debidas al uso del oscilador externo y cambio de frecuencia, debiendo de hacer cambios a los parámetros de tiempo a cada módulo de ser necesario.
- Respecto al punto anterior se debe realizar una recalibración del ADC ajustando los factores de conversión que se encuentran en el archivo MAIN. Por otro lado, el baudrate del sistema de comunicación debe ser medido, dado que el sistema no comparte el reloj de comunicación se debe corroborar de que el sistema de control externo y celda se comunican con la tasa de transmisión fijada de forma correcta. Al tener en funcionamiento el sistema de comunicación y adquisición de datos posibilita la puesta en prueba del módulo de disparos y posterior cierre del lazo de control.
- El sistema de conversión de datos utiliza módulos multiplicadores que utilizan una gran cantidad de recursos de la FPGA, utilizar la implementación basada en RAMs es una opción viable para las multiplicaciones que tienen un factor constante en caso de requerir más espacio.

Bibliografía

[1] S. Kouro, J. Rodriguez, B. Wu, S. Bernet, and M. Perez, "Powering the future of industry: High-power adjustable speed drive topologies," *IEEE Industry Applications Magazine*, vol. 18, no. 4, pp. 26–39, 2012. 2.1

[2] S. Bhattacharya, "Transforming the transformer," *IEEE Spectrum*, vol. 54, no. 7, pp. 38–43, 2017. 2.1

[3] X. She, A. Q. Huang, and R. Burgos, "Review of solid-state transformer technologies and their application in power distribution systems," *IEEE Journal of Emerging and Selected Topics in Power Electronics*, vol. 1, no. 3, pp. 186–198, 2013. 2.1

[4] W. McMurray, "The thyristor electronic transformer: a power converter using a high-frequency link," *IEEE Transactions on Industry and General Applications*, vol. IGA-7, no. 4, pp. 451–457, 1971. 2.1

[5] A. Q. Huang, Q. Zhu, L. Wang, and L. Zhang, "15 kv sic mosfet: An enabling technology for medium voltage solid state transformers," *CPSS Transactions on Power Electronics and Applications*, vol. 2, no. 2, pp. 118–130, 2017. 2.1

[6] F. Ruiz, M. A. Perez, J. R. Espinosa, T. Gajowik, S. Stynski, and M. Malinowski, "Surveying solid-state transformer structures and controls: Providing highly efficient and controllable power flow in distribution grids," *IEEE Industrial Electronics Magazine*, vol. 14, no. 1, pp. 56–70, 2020. 2.1, 2.1, 2.1

[7] C. Zhao, D. Dujic, A. Mester, J. K. Steinke, M. Weiss, S. Lewdeni-Schmid, T. Chaudhuri, and P. Stefanutti, "Power electronic traction transformer—medium voltage prototype," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 7, pp. 3257–3268, 2014. 2.1

[8] D. Dujic, G. K. Steinke, M. Bellini, M. Rahimo, L. Storasta, and J. K. Steinke, "Characterization of 6.5 kv igbts for high-power medium-frequency soft-switched applications," *IEEE Transactions on Power Electronics*, vol. 29, no. 2, pp. 906–919, 2014. 2.1, 2.1

[9] M. Hiller, S. Busse, and A.-H. Gheeth, "Modular multilevel converter (M2C) Medium Voltage Drives," Siemens, Technical report, 2016. 2.1

[10] P. Services, M.-s. Solutions, B. technology, D. Technology, I. Automation, C. Products, F. Couplings, F. Units, G. Motors, and S. e. a. Tools, "Sinamics medium voltage drives," 2020. [Online]. Available: https://new.siemens.com/global/en/products/drives/sinamics/medium-voltage-converters.html 2.1

[11] P. Himmelmann, M. Hiller, D. Krug, and M. Beuermann, "A new modular multilevel converter for medium voltage high power oil gas motor drive applications," in *2016 18th European Conference on Power Electronics and Applications (EPE'16 ECCE Europe)*, 2016, pp. 1–11. 2.1

[12] S. Busse, M. Hiller, K. Kahlen, and P. Himmelmann, "Mtbf comparison of cutting edge medium voltage drive topologies for oil gas applications," in *2015 Petroleum and Chemical Industry Conference Europe (PCIC Europe)*, 2015, pp. 1–13. 2.1

[13] D. Yadav and A. Singh, *Microcontrollers: Features and Applications*. New Age International (P) Limited Publishers, 2006. [Online]. Available: https://books.google.cl/books?id=SHM_eMv2ZqYC 2.2.1

[14] A. Trevennor, *A Brief History of Microcontrollers*. Berkeley, CA: Apress, 2012, pp. 3–11. [Online]. Available: https://doi.org/10.1007/978-1-4302-4447-9_1 2.2.1

[15] "Microchip Microcontrollers," 2020. [Online]. Available: https://www.microchip.com/design-centers/microcontrollers 2.2.1

[16] "TI Microcontrollers," 2020. [Online]. Available: https://www.ti.com/microcontrollers/overview.html 2.2.1

[17] "NXP Microcontrollers," 2020. [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers:MICROCONTROLLERS-AND-PROCESSORS#/ 2.2.1

[18] M. Corture, "History and future of digital signal processors," 2020. [Online]. Available: https://www.curtisswrightds.com/news/blog/history-and-future-of-digital-signal-processors.html 2.2.2

[19] J. M. Peinado, "Estructura interna y funcionamiento de periféricos en sistemas microprocesadores. desarrollo de una aplicación multimedia en formato cd-rom," Ph.D. dissertation, Universidad de Sevilla, 2008. 2.2.2

[20] "Microchip DSPic," 2020. [Online]. Available: https://www.microchip.com/design-centers/motor-control-and-drive/motor-control-products/dspic33-digital-signal-controllers 2.2.2

[21] "NXP DSP," 2020. [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/additional-architectures/digital-signal-processors:Digital-Signal-Processors 2.2.2

[22] "TI DSP," 2020. [Online]. Available: https://www.microchip.com/design-centers/motor-control-and-drive/motor-control-products/dspic33-digital-signal-controllers 2.2.2

[23] D. Romano, "A brief history of fpga," 2020. [Online]. Available: https://makezine.com/2019/10/11/a-brief-history-of-fpga/ 2.2.3

[24] H. Sakulin, "Introduction to field programmable gate arrays," CERN / EP-CMD, 8th International School for Trigger and Data Acquisition, 2017. 2.2.3

[25] "Fpga vs Microcontroller which is better for your needs," 2020. [Online]. Available: https://www.ourpcb.com/fpga-vs-microcontroller.html 2.2.3

[26] *SPI Interface*, Analog Devices, 2015, rev. A. 2.3.1

[27] *Serial Peripheral Interface*, Microchip Technology, 2011, rev. G. 2.3.1

[28] *I2C-bus specification and user manual*, NXP Semiconductors, 2014, rev. 6. 2.3.2

[29] *KeyStone Architecture Universal Asynchronous Receiver/Transmitter (UART)*, Texas Instrument, 2010, rev. 1. 2.3.3

[30] *USART*, Microchip Technology, 2008, rev. 1. 2.3.4

[31] B. Biswas and B. Chaudhur, "Generalization of booth's algorithm for efficient multiplication," *Procedia Technology*, vol. 10, pp. 304 – 310, 2013. 4.6

[32] G. W. Bewick, "Fast multiplication: Algorithms and implementation," Ph.D. dissertation, Standford University, 1994. 4.6

A ANEXO 1

Es de importancia que las señales que comandan los semiconductores de potencia estén definidas correctamente, dado que el trabajo de la placa asociada del hardware de potencia fue desarrollado por otro equipo, las etiquetas que nombran las señales no son iguales en la Figura A.1 se muestra la relación entre los nombres que referencian a las señales mencionadas.

Por otro lado, las señales mencionadas están simbolizadas internamente en el código, los disparos correspondiente a cada IGBT enumerado se encuentra representado en las Figuras A.2 y A.3.

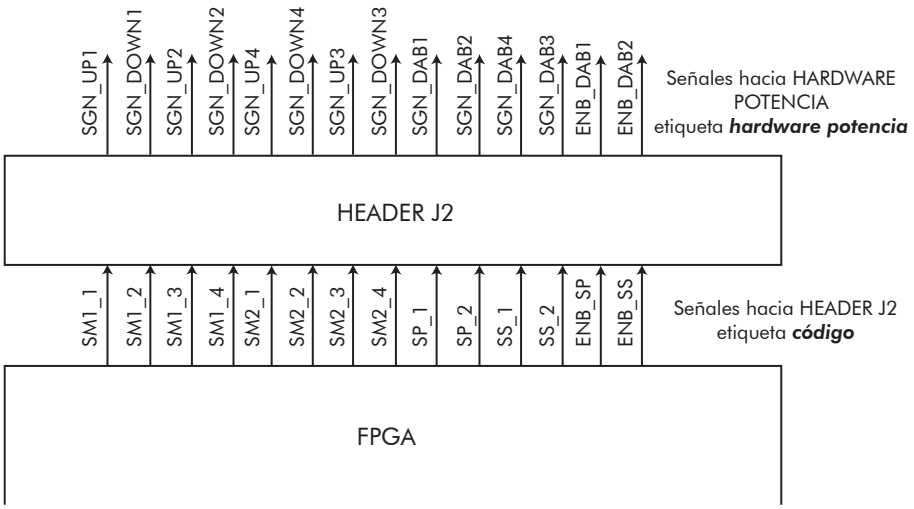


Figura A.1: Etiqueta de señales de semiconductores

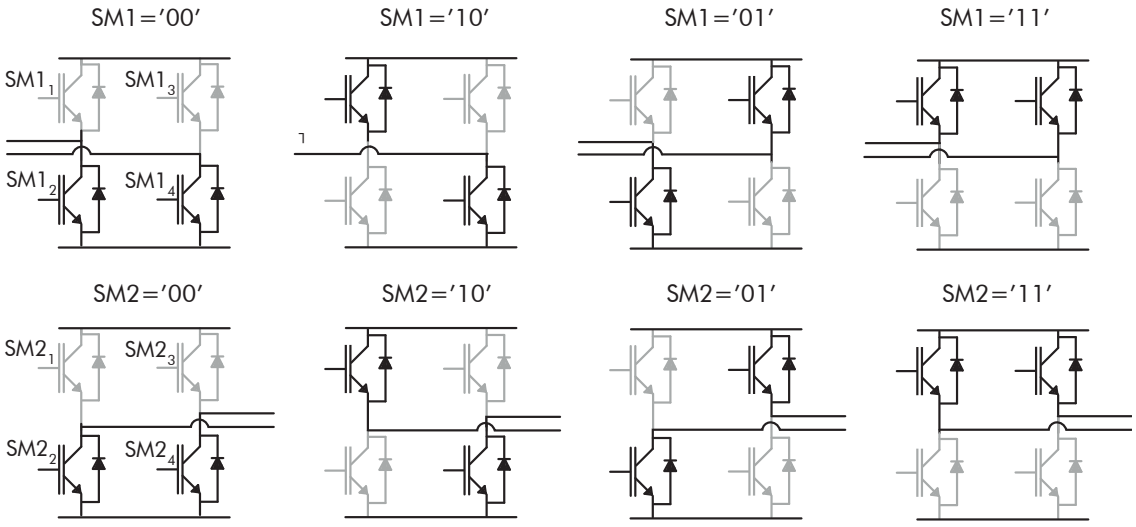


Figura A.2: Semiconductores encendidos por señal de control SM1 y SM2

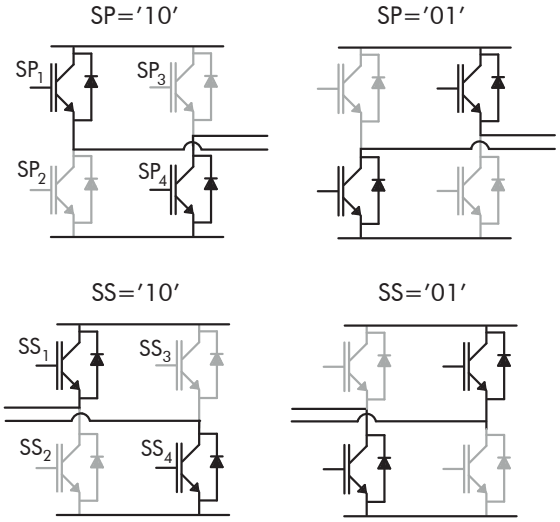


Figura A.3: Semiconductores encendidos por señal de control SP y SS

B ANEXO 2

Códigos VHDL

B.1. Main

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.STD_LOGIC_ARITH.all;
4  use IEEE.NUMERIC_STD.all;
5  library MACHXO2;
6  use MACHXO2.COMPONENTS.all;
7
8  entity MAIN is
9        port (
10                          --CLK: IN STD_LOGIC; USA EL OSCILADOR EXTERNO
11                          RST: in STD_LOGIC;
12
13                          RX_INPUT: in STD_LOGIC;
14
15                          CLK_OUT_ADC: in STD_LOGIC;
16                          SD1: in STD_LOGIC;
17                          SD2: in STD_LOGIC;
18                          SD3: in STD_LOGIC;
19
20                          TX_OUTPUT : out STD_LOGIC;
21
22
23                          CNV: out STD_LOGIC;
24                          SCK: out STD_LOGIC;
25
26                          SM1_1: out STD_LOGIC;
27                          SM1_2: out STD_LOGIC;
28                          SM1_3 : out STD_LOGIC;
29                          SM1_4 : out STD_LOGIC;
30                          SM2_1 : out STD_LOGIC;
31                          SM2_2 : out STD_LOGIC;
32                          SM2_3 : out STD_LOGIC;
33                          SM2_4 : out STD_LOGIC;
34                          SP_EN : out STD_LOGIC;
35                          SS_EN : out STD_LOGIC;
36                          SP_1 : out STD_LOGIC;
37                          SP_3 : out STD_LOGIC;
38                          SS_1 : out STD_LOGIC;
39                          SS_3 : out STD_LOGIC;
40
41                          OP_OK1 : out STD_LOGIC;
42                          EM_ON1 : out STD_LOGIC
43                         );
44 end MAIN;
45
```

```
46  architecture ARCH of MAIN is
47
48  signal CLK: STD_LOGIC;
49
50
51
52  signal W2S: STD_LOGIC_VECTOR(8 downto 0);
53  signal W2W: STD_LOGIC_VECTOR(7 downto 0);
54  signal MSG_SENT: STD_LOGIC;
55  signal MSG_TRUE: STD_LOGIC;
56  signal BUSY_RX: STD_LOGIC;
57  signal BUSY_TX: STD_LOGIC;
58  signal READTICK: STD_LOGIC;
59
60
61
62  ---- SEÑALES DECODIFICADOR ----
63  signal                VC1_REF : STD_LOGIC_VECTOR(1 downto 0);
64  signal                VC2_REF : STD_LOGIC_VECTOR(1 downto 0);
65  signal                M_OP: STD_LOGIC_VECTOR(1 downto 0);
66  signal                SM1 : STD_LOGIC_VECTOR(1 downto 0);
67  signal                SM2 : STD_LOGIC_VECTOR(1 downto 0);
68  signal                I_REF: STD_LOGIC_VECTOR(11 downto 0);
69  signal                EN_SM1 : STD_LOGIC;
70  signal                EN_SM2 : STD_LOGIC;
71  ---- SEÑALES CODIFICADOR ---
72  signal V_C1 : STD_LOGIC_VECTOR(11 downto 0);
73  signal V_C2 : STD_LOGIC_VECTOR(11 downto 0);
74
75  --- SEÑALES UNIDAD DE CONTROL ----
76  signal OP_OK: STD_LOGIC;
77  signal EM_ON: STD_LOGIC;
78
79  ---- SEÑALES INTERFAZ ----
80  signal ADC_OK:STD_LOGIC;
81
82
83  --- SEÑALES LAZO CONTROL
84  signal OVER_VOLTAGE: STD_LOGIC;
85  signal OVER_CURRENT: STD_LOGIC;
86  signal                DUTY_PWM_SP : INTEGER:=18000;
87  signal                DUTY_PWM_SS : INTEGER:=6000;
88  signal                PHASE_PWM_SS : INTEGER:=4500;
89
90  ---- SEÑALES INTERFAZ ----
91  signal                VC1: STD_LOGIC_VECTOR(15 downto 0);
92  signal                VC2: STD_LOGIC_VECTOR(15 downto 0);
93  signal                IP:  STD_LOGIC_VECTOR(15 downto 0);
94
95
96
97  constant FACTOR_VOLTAGE: STD_LOGIC_VECTOR(15 downto 0):="0000000010010110";
98  constant FACTOR_CORRIENTE: STD_LOGIC_VECTOR(15 downto 0):="0000000000000101";
99  constant OFFSET_CORRIENTE: STD_LOGIC_VECTOR(15 downto 0):="0000001000000000";
100 constant VOLTAJE_MAXIMO: STD_LOGIC_VECTOR(15 downto 0):="0000001110000011";
101 ---- Q6.10 (450/512) SIGNO
102 constant CORRIENTE_MAXIMA: STD_LOGIC_VECTOR(15 downto 0):="0000001100000000";
103 ---- Q6.10 (12/16) SIGNO
104 constant CLK_EN:STD_LOGIC:='1';
105 constant TX_FLAG: STD_LOGIC:='1';
106
107
108
109 signal V_C1Q: STD_LOGIC_VECTOR(15 downto 0);
110 signal V_C1QQ: STD_LOGIC_VECTOR(31 downto 0);
111 signal V_C1A: STD_LOGIC_VECTOR(15 downto 0);
112 signal V_C2Q: STD_LOGIC_VECTOR(15 downto 0);
113 signal V_C2QQ: STD_LOGIC_VECTOR(31 downto 0);
```

```
114 signal V_C2A: STD_LOGIC_VECTOR(15 downto 0);
115 signal IPQ: STD_LOGIC_VECTOR(15 downto 0);
116 signal IPQQ: STD_LOGIC_VECTOR(31 downto 0);
117 signal IPQQQ: STD_LOGIC_VECTOR(15 downto 0);
118 signal IPA: STD_LOGIC_VECTOR(15 downto 0);
119
120 ----- COMPONENTES
121
122 -- COMENTAR DE USAR EL OSCILADOR EXTERNO
123    component OSCH
124       generic(
125             NOM_FREQ: STRING := "2.08");
126       port(
127             STDBY                    : in STD_LOGIC;
128             OSC                             : out STD_LOGIC;
129             SEDSTDBY        : out STD_LOGIC);
130    end component;
131 ----------------------------------------
132
133 component TX_UART
134          port(
135                 CLK : in STD_LOGIC;  -- FPGA M_CLK
136                 RESET :in STD_LOGIC;  -- RESET GENERAL
137                 TX_FLAG: in STD_LOGIC;         -- BANDERA PARA SOLICITAR EL MÓDULO
138                 WORD_TO_SEND : in STD_LOGIC_VECTOR(8 downto 0);        -- PALABRA A ESCRIBIR
139                 OUTPUT : out STD_LOGIC;          -- BIT A ESCRIBIR
140                 MSG_SENT :out STD_LOGIC;  -- BANDERA DE MENSAJE ENVIADO
141                 BUSY_TX :out STD_LOGIC                  -- BANDERA DE CANAL OCUPADO
142              );
143 end component;
144
145 component RX_UART
146          port(
147          CLK : in STD_LOGIC;
148          RESET :in STD_LOGIC;
149          INPUT: in STD_LOGIC;
150          READTICK: out STD_LOGIC;
151          WORD_RECEIVED : out STD_LOGIC_VECTOR(7 downto 0);
152          BUSY_RX :out STD_LOGIC;
153          MSG_TRUE: out STD_LOGIC
154              );
155 end component;
156
157 component DECODIFICADOR_DSPACE
158         port(
159                 clk : in STD_LOGIC;
160                 reset :in std_logic;
161                 msg_true :in std_logic;
162                 rx_word : in std_logic_vector(7 downto 0);
163                 VC1_ref : out std_logic_vector(1 downto 0);
164                 VC2_ref : out std_logic_vector(1 downto 0);
165                 M_op: out std_logic_vector(1 downto 0);
166                 SM1 : out std_logic_vector(1 downto 0);
167                 SM2 : out std_logic_vector(1 downto 0);
168                 EN_SM1 : out std_logic;
169                 EN_SM2 : out std_logic;
170                 i_ref: out std_logic_vector(11 downto 0)
171                 );
172 end component;
173
174 component CODIFICADOR_DSPACE is
175         port(
176                 CLK : in STD_LOGIC;
177                 RESET :in STD_LOGIC;
178                 BUSY_TX: in STD_LOGIC;
179                 ADC_OK:        in STD_LOGIC;
180                 V_C1 : in STD_LOGIC_VECTOR(11 downto 0);
181                 V_C2 : in STD_LOGIC_VECTOR(11 downto 0);
```

```
182                 MSG_SENT : in STD_LOGIC;
183                 WORD_TO_SEND: out STD_LOGIC_VECTOR(8 downto 0)
184                 );
185 end component;
186
187 component DISPARADOR is
188         port(
189                 CLK : in STD_LOGIC;
190                 RESET :in STD_LOGIC;
191                 OP_OK :in STD_LOGIC;
192                 SM1 : in STD_LOGIC_VECTOR(1 downto 0);
193                 SM2 : in STD_LOGIC_VECTOR(1 downto 0);
194                 DUTY_PWM_SP : in INTEGER;
195                 DUTY_PWM_SS : in INTEGER;
196                 PHASE_PWM_SS : in INTEGER;
197                 EN_SM1 :in std_logic;
198                 EN_SM2 :in std_logic;
199                 SM1_1 :out STD_LOGIC;
200                 SM1_2 :out STD_LOGIC;
201                 SM1_3 :out STD_LOGIC;
202                 SM1_4 :out STD_LOGIC;
203                 SM2_1 :out STD_LOGIC;
204                 SM2_2 :out STD_LOGIC;
205                 SM2_3 :out STD_LOGIC;
206                 SM2_4 :out STD_LOGIC;
207                 SP_EN :out STD_LOGIC;
208                 SS_EN :out STD_LOGIC;
209                 SP_1 :out STD_LOGIC;
210                 SP_3 :out STD_LOGIC;
211                 SS_1 :out STD_LOGIC;
212                 SS_3 :out STD_LOGIC
213                                 );
214 end component;
215
216 component UNIDAD_DE_CONTROL_CENTRAL is
217         port(
218                 CLK : in STD_LOGIC;
219                 RESET :in STD_LOGIC;
220                 M_OP :in STD_LOGIC_VECTOR(1 downto 0);
221                 OVER_VOLTAGE :in STD_LOGIC;
222                 OVER_CURRENT :in STD_LOGIC;
223                 OP_OK: out STD_LOGIC;
224                 EM_ON: out STD_LOGIC
225                 );
226 end component;
227
228 component LAZO_DE_CONTROL is
229         port(
230                 CLK : in STD_LOGIC;
231                 RESET :in STD_LOGIC;
232                 M_OP :in STD_LOGIC_VECTOR(1 downto 0);
233                 VC1_REF : in STD_LOGIC_VECTOR(1 downto 0);
234                 VC2_REF : in STD_LOGIC_VECTOR(1 downto 0);
235                 I_REF: in STD_LOGIC_VECTOR(11 downto 0);
236                 VC1: in STD_LOGIC_VECTOR(11 downto 0);
237                 VC2: in STD_LOGIC_VECTOR(11 downto 0);
238                 IP: in STD_LOGIC_VECTOR(11 downto 0);
239                 OP_OK: in STD_LOGIC;
240                 DUTY_PWM_SP : out INTEGER;
241                 DUTY_PWM_SS : out INTEGER;
242                 PHASE_PWM_SS : out INTEGER
243                 );
244 end component;
245
246 component INTERFAZ_ADC is
247         port(
248                 CLK : in STD_LOGIC;
249                 RESET :in STD_LOGIC;
```

```
250                 CLK_OUT_ADC :in STD_LOGIC;
251                 SD1: in STD_LOGIC;
252                 SD2: in STD_LOGIC;
253                 SD3: in STD_LOGIC;
254                 CNV: out STD_LOGIC;
255                 SCK: out STD_LOGIC;
256                 VC1: out STD_LOGIC_VECTOR(15 downto 0);
257                 VC2: out STD_LOGIC_VECTOR(15 downto 0);
258                 IP: out STD_LOGIC_VECTOR(15 downto 0);
259                 ADC_OK:        out STD_LOGIC
260                 );
261 end component;
262
263 component VOLTAJE1 is
264         port(
265         CLOCK: in  STD_LOGIC;
266         CLKEN: in  STD_LOGIC;
267         ACLR: in  STD_LOGIC;
268         DATAA: in  STD_LOGIC_VECTOR(15 downto 0);
269         DATAB: in  STD_LOGIC_VECTOR(15 downto 0);
270         RESULT: out  STD_LOGIC_VECTOR(31 downto 0)
271                 );
272 end component;
273
274 component VOLTAJE2 is
275         port(
276         CLOCK: in  STD_LOGIC;
277         CLKEN: in  STD_LOGIC;
278         ACLR: in  STD_LOGIC;
279         DATAA: in  STD_LOGIC_VECTOR(15 downto 0);
280         DATAB: in  STD_LOGIC_VECTOR(15 downto 0);
281         RESULT: out  STD_LOGIC_VECTOR(31 downto 0)
282                 );
283 end component;
284
285 component CORRIENTE is
286         port(
287         CLOCK: in  STD_LOGIC;
288         CLKEN: in  STD_LOGIC;
289         ACLR: in  STD_LOGIC;
290         DATAA: in  STD_LOGIC_VECTOR(15 downto 0);
291         DATAB: in  STD_LOGIC_VECTOR(15 downto 0);
292         RESULT: out  STD_LOGIC_VECTOR(31 downto 0)
293                 );
294 end component;
295
296
297 component CORRIENTE_SUM is
298     port (
299         DATAA: in  STD_LOGIC_VECTOR(15 downto 0);
300         DATAB: in  STD_LOGIC_VECTOR(15 downto 0);
301         RESULT: out  STD_LOGIC_VECTOR(15 downto 0));
302 end component;
303 -------------------------------------------------
304 -------------------------------------------------
305 -------------------------------------------------
306
307 begin
308
309 --COMENTAR DE USAR EL OSCILADOR EXTERNO-------------------------
310    OSCINST0: OSCH
311       generic map (NOM_FREQ  => "2.08")
312       port map (STDBY => '0', OSC => CLK,  SEDSTDBY => open);
313 ---------------------------------------------------------------
314
315         C1: TX_UART port map (CLK,RST,TX_FLAG,W2S,TX_OUTPUT,MSG_SENT,BUSY_TX);
316         C2: RX_UART port map (CLK,RST,RX_INPUT,READTICK,W2W,BUSY_RX,MSG_TRUE);
317         C3: DECODIFICADOR_DSPACE port map (CLK,RST,MSG_TRUE,W2W,VC1_REF,VC2_REF,M_OP,SM1,SM2,
```

```
318         EN_SM1,EN_SM2,I_REF);
319         C4: CODIFICADOR_DSPACE port map (CLK,RST,BUSY_TX,ADC_OK,V_C1,V_C2,MSG_SENT,W2S);
320         C5: DISPARADOR port map (CLK,RST,OP_OK,SM1,SM2,DUTY_PWM_SP,DUTY_PWM_SS,
321         PHASE_PWM_SS,EN_SM1,EN_SM2,SM1_1,SM1_2,SM1_3,SM1_4,SM2_1,SM2_2,SM2_3,SM2_4,
322         SP_EN,SS_EN,SP_1,SP_3,SS_1,SS_3);
323         C6: UNIDAD_DE_CONTROL_CENTRAL port map (CLK,RST,M_OP,OVER_VOLTAGE,OVER_CURRENT,OP_OK,EM_ON);
324         C7: INTERFAZ_ADC port map (CLK,RST,CLK_OUT_ADC,SD1,SD2,SD3,CNV,SCK,VC1,VC2,IP,ADC_OK);
325         C9: VOLTAJE1 port map (CLK,CLK_EN,RST,V_C1Q,FACTOR_VOLTAGE,V_C1QQ);
326         C10: VOLTAJE2 port map (CLK,CLK_EN,RST,V_C2Q,FACTOR_VOLTAGE,V_C2QQ);
327         C11: CORRIENTE port map (CLK,CLK_EN,RST,IPQ,FACTOR_CORRIENTE,IPQQ);
328         C12: CORRIENTE_SUM port map (IPQQQ,OFFSET_CORRIENTE,IPA);
329         ----> IPA ENTREGA LA CORRIENTE QUE DEBE ENTRAR AL LAZO DE CONTROL
330
331
332
333 ---- ASIGNACIÓN DE REGISTROS  DE DATOS -------------
334 ------------------------------------------------
335 process (VC1)
336 begin
337         V_C1<=VC1(14 downto 3);
338         V_C1Q<="00" & VC1(15 downto 2); --MULTIPLICA POR 2 LA MEDICION
339 end process;
340
341 process (VC2)
342 begin
343         V_C2<=VC2(14 downto 3);
344         V_C2Q<="00" & VC2(15 downto 2) ; --MULTIPLICA POR 2 LA MEDICION
345 end process;
346
347 process (V_C1QQ)
348 begin
349         V_C1A<=V_C1QQ(25 downto 10); -- RESULTADO DE VQ*0,14652/512 ---> V_PU
350 end process;
351
352 process (V_C2QQ)
353 begin
354         V_C2A<=V_C2QQ(25 downto 10); -- RESULTADO DE VQ*0,14652/512 ---> V_PU
355 end process;
356
357
358 process (IP)
359 begin
360         IPQ<="000" & IP(15 downto 3); -- TOMA LOS 12 BITS + SIGNO DE LA MEDICION
361 end process;
362
363 process (IPQQ)
364 begin
365         IPQQQ<=IPQQ(19 downto 4); -- RESULTADO DE IQ*0.005/16
366 end process;
367
368
369
370 ------------------------------------------------
371 ------------------------------------------------
372
373
374 --------------- LÓGICA DE EMERGENCIA -----------
375 process(RST,V_C1A,V_C2A)
376 begin
377         if (RST='1') then
378         OVER_VOLTAGE<='0';
379         elsif (V_C1A>VOLTAJE_MAXIMO or V_C2A>VOLTAJE_MAXIMO) then
380         OVER_VOLTAGE<='1';
381         else
382         OVER_VOLTAGE<='0';
383         end if;
384 end process;
385
```

```
386
387
388 process(RST,IPA)
389 begin
390         if (RST='1') then
391         OVER_CURRENT<='0';
392         elsif (IPA>CORRIENTE_MAXIMA) then
393         OVER_CURRENT<='1';
394         else
395         OVER_CURRENT<='0';
396         end if;
397 end process;
398
399 --------------------
400
401 process(OP_OK,EM_ON)
402 begin
403         OP_OK1<=OP_OK;
404         EM_ON1<=EM_ON;
405 end process;
406
407  end ARCH;
```

B.2. TX UART

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3
4  entity TX_UART is
5          port(
6                  CLK : in STD_LOGIC;  -- FPGA M_CLK
7                  RESET :in STD_LOGIC;  -- RESET GENERAL
8                  TX_FLAG: in STD_LOGIC;         -- BANDERA PARA SOLICITAR EL MÓDULO
9                  WORD_TO_SEND : in STD_LOGIC_VECTOR(8 downto 0);        -- PALABRA A ESCRIBIR
10                 OUTPUT : out STD_LOGIC;          -- BIT A ESCRIBIR
11                 MSG_SENT :out STD_LOGIC;  -- BANDERA DE MENSAJE ENVIADO
12                 BUSY_TX :out STD_LOGIC                  -- BANDERA DE CANAL OCUPADO
13                 );
14 end TX_UART;
15
16
17 architecture BEHAVIORAL of TX_UART is
18         type ESTADOS is (IDLE,FB,S1,S2,S3,S4,S5,S6,S7,S8,STOP,STOP2); -- DEFINICIÓN DE LOS ESTADOS DE LA MÁQUINA
19         signal REG_ESTADO,ESTADO_SIGUIENTE:ESTADOS;        --DECLARACIÓN DEL VECTOR DE ESTADOS
20         signal TX_WORD: STD_LOGIC_VECTOR(7 downto 0); -- PALABRA A ESCRIBIR
21         signal WRITE: STD_LOGIC;  --BANDERA PARA ESCRIBIR EL BIT DEL MENSAJE
22
23
24 begin
25         --TRANSICION DE ESTADOS---
26         process (CLK, RESET)
27         begin
28                 if RESET='1' then
29                         REG_ESTADO<=IDLE;
30
31
32                 elsif (CLK'EVENT and CLK='1') then
33                         REG_ESTADO <= ESTADO_SIGUIENTE;
34                 end if;
35         end process;
36
37         --LOGICA DE ESTADOS---
38         process(REG_ESTADO,WRITE,TX_FLAG,WORD_TO_SEND)
39         begin
40                 ESTADO_SIGUIENTE <= REG_ESTADO;
```

```
41                 case (REG_ESTADO) is
42                         when IDLE =>
43                                 if (WRITE='1' and TX_FLAG='1' and WORD_TO_SEND(8)='1') then
44                                         ESTADO_SIGUIENTE <= FB;
45
46                                 else
47                                         ESTADO_SIGUIENTE <= IDLE;
48                         end if;
49                         when FB =>
50                                 if (WRITE='1') then
51                                         ESTADO_SIGUIENTE <= S1;
52                                 else
53                                         ESTADO_SIGUIENTE <= FB;
54                                 end if;
55
56                         when S1 =>
57                                 if (WRITE='1') then
58                                         ESTADO_SIGUIENTE <= S2;
59                                 else
60                                         ESTADO_SIGUIENTE <= S1;
61                         end if;
62                         when S2 =>
63                                 if (WRITE='1')        then
64                                         ESTADO_SIGUIENTE <= S3;
65                                 else
66                                         ESTADO_SIGUIENTE <= S2;
67                                 end if;
68
69                         when S3 =>
70                                 if (WRITE='1') then
71                                         ESTADO_SIGUIENTE <= S4;
72                                 else
73                                         ESTADO_SIGUIENTE <= S3;
74                         end if;
75                         when S4 =>
76                                 if (WRITE='1')        then
77                                         ESTADO_SIGUIENTE <= S5;
78                                 else
79                                         ESTADO_SIGUIENTE <= S4;
80                                 end if;
81
82                         when S5 =>
83                                 if (WRITE='1') then
84                                         ESTADO_SIGUIENTE <= S6;
85                                 else
86                                         ESTADO_SIGUIENTE <= S5;
87                         end if;
88                         when S6 =>
89                                 if (WRITE='1')        then
90                                         ESTADO_SIGUIENTE <= S7;
91                                 else
92                                         ESTADO_SIGUIENTE <= S6;
93                         end if;
94                         when S7 =>
95                                 if (WRITE='1') then
96                                         ESTADO_SIGUIENTE <= S8;
97                                 else
98                                         ESTADO_SIGUIENTE <= S7;
99                         end if;
100                        when S8 =>
101                                if (WRITE='1')        then
102                                        ESTADO_SIGUIENTE <= STOP;
103                                else
104                                        ESTADO_SIGUIENTE <= S8;
105                                end if;
106
107                        when STOP =>
108                                if (WRITE='1')        then
```

```
109                                     ESTADO_SIGUIENTE <= STOP2;
110                             else
111                                     ESTADO_SIGUIENTE <= STOP;
112                     end if;
113                     when STOP2 =>
114                     ESTADO_SIGUIENTE <= IDLE;
115             end case;
116     end process;
117
118
119     process(REG_ESTADO,WORD_TO_SEND)
120     begin
121             case(REG_ESTADO) is
122                     when IDLE => OUTPUT <= '1';
123                             BUSY_TX <= '0';
124                     MSG_SENT <='0';
125                     when FB        => OUTPUT <= '0';
126                             BUSY_TX <= '1';
127                     MSG_SENT <='0';
128                     when S1 => OUTPUT <= WORD_TO_SEND(0);
129                             BUSY_TX <= '1';
130                     MSG_SENT <='0';
131                     when S2        => OUTPUT <= WORD_TO_SEND(1);
132                             BUSY_TX <= '1';
133                     MSG_SENT <='0';
134                     when S3 => OUTPUT <= WORD_TO_SEND(2);
135                             BUSY_TX <= '1';
136                     MSG_SENT <='0';
137                     when S4        => OUTPUT <= WORD_TO_SEND(3);
138                             BUSY_TX <= '1';
139                     MSG_SENT <='0';
140                     when S5 => OUTPUT <= WORD_TO_SEND(4);
141                             BUSY_TX <= '1';
142                     MSG_SENT <='0';
143                     when S6        => OUTPUT <= WORD_TO_SEND(5);
144                             BUSY_TX <= '1';
145                     MSG_SENT <='0';
146                     when S7 => OUTPUT <= WORD_TO_SEND(6);
147                             BUSY_TX <= '1';
148                     MSG_SENT <='0';
149                     when S8        => OUTPUT <= WORD_TO_SEND(7);
150                             BUSY_TX <= '1';
151                     MSG_SENT <='0';
152                     when STOP        => OUTPUT <= '1';
153                             BUSY_TX <= '1';
154                             MSG_SENT <='0';
155
156                     when STOP2        => OUTPUT <= '1';
157                             BUSY_TX <= '1';
158                             MSG_SENT <='1';
159
160             end case;
161     end process;
162
163
164     process(CLK,REG_ESTADO)
165             variable CONTADOR_TXA : INTEGER range 0 to 1000;
166     begin
167             if (RESET='1') then
168             CONTADOR_TXA := 0;
169             WRITE<='0';
170             elsif (RISING_EDGE(CLK)) then
171                     if (CONTADOR_TXA < 9) then  --CONTADOR=(MASTER_CLK/BAUDRATE)-1
172                             CONTADOR_TXA := CONTADOR_TXA+1;
173                             WRITE<='0';
174                     elsif(CONTADOR_TXA < 14 and REG_ESTADO=STOP) then
175
176                     --CONTADOR=(MASTER_CLK/BAUDRATE-1)*1.5 STOP BIT CONDITION
```

```
177                                 CONTADOR_TXA := CONTADOR_TXA+1;
178                                 WRITE<='0';
179                             else
180                                 CONTADOR_TXA := 0;
181                                 WRITE<='1';
182                             end if;
183                     end if;
184             end process;
185
186     end BEHAVIORAL;
```

B.3. RX UART

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.STD_LOGIC_ARITH.all;
4  use IEEE.STD_LOGIC_UNSIGNED.all;
5  use IEEE.NUMERIC_STD;
6
7  entity RX_UART is
8           port(
9           CLK : in STD_LOGIC;
10          RESET :in STD_LOGIC;
11          INPUT: in STD_LOGIC;
12          READTICK: out STD_LOGIC;
13          WORD_RECEIVED : out STD_LOGIC_VECTOR(7 downto 0);
14          BUSY_RX :out STD_LOGIC;
15          MSG_TRUE: out STD_LOGIC
16
17              );
18 end RX_UART;
19
20
21 architecture BEHAVIORAL of RX_UART is
22 type ESTADOS is (IDLE,FB,S1,STOP);
23 signal REG_ESTADO,ESTADO_SIGUIENTE:ESTADOS;
24 signal CONTADOR_BIT: INTEGER range 0 to 10;
25 signal N_CONTADOR_BIT: INTEGER range 0 to 10;
26 signal RX_FLAG: STD_LOGIC;
27 signal STORED_WORD : STD_LOGIC_VECTOR(7 downto 0);
28 signal N_STORED_WORD : STD_LOGIC_VECTOR(7 downto 0);
29 signal READ: STD_LOGIC;
30
31 begin
32 --TRANSICION DE ESTADOS---
33 process (CLK, RESET)
34 begin
35         if RESET='1' then
36                 REG_ESTADO<=IDLE;
37
38
39         elsif (CLK'EVENT and CLK='1') then
40                 REG_ESTADO <= ESTADO_SIGUIENTE;
41                 CONTADOR_BIT <= N_CONTADOR_BIT;
42                 STORED_WORD<=N_STORED_WORD;
43
44         end if;
45 end process;
46
47 --MAQUINA DE ESTADOS---
48 process(all)
49 begin
50         ESTADO_SIGUIENTE <= REG_ESTADO;
51         N_CONTADOR_BIT<=CONTADOR_BIT;
52         N_STORED_WORD<=STORED_WORD;
53
54                 case (REG_ESTADO) is
55                 when IDLE =>
56                                 ----SALIDAS----
57                                 RX_FLAG<='0';
58                                 MSG_TRUE<='0';
59                                 N_STORED_WORD<="00000000";
60                                 N_CONTADOR_BIT<=0;
61                                 WORD_RECEIVED<= "00000000";
62                                 ----LOGICA DEL SIGUIENTE ESTADO----
63                                 if (INPUT='0') then
64                                         ESTADO_SIGUIENTE <= FB;
65                                 else
```

```
66                                      ESTADO_SIGUIENTE <= IDLE;
67                              end if;
68                  when FB =>
69                          ----SALIDAS----
70                                  RX_FLAG<='1';
71                                  MSG_TRUE<='0';
72                                  N_STORED_WORD<="00000000";
73                                  N_CONTADOR_BIT<=0;
74                                  WORD_RECEIVED<= "00000000";
75                                  ----LOGICA DEL SIGUIENTE ESTADO----
76                                  if (READ='1' and INPUT='1') then
77                                          ESTADO_SIGUIENTE <= IDLE;
78                                  elsif (READ='1' and INPUT='0') then
79                                          ESTADO_SIGUIENTE <= S1;
80                                  else
81                                          ESTADO_SIGUIENTE <= FB;
82                                  end if;
83
84
85                   when S1 =>
86                                   ----SALIDAS----
87                                  RX_FLAG<='1';
88                                  MSG_TRUE<='0';
89                                  WORD_RECEIVED<= "00000000";
90                                   if (READ='1') then
91                                           N_STORED_WORD(CONTADOR_BIT)<= INPUT;
92                                           N_CONTADOR_BIT<=CONTADOR_BIT+1;
93                                   else
94                                          N_CONTADOR_BIT<=CONTADOR_BIT;
95                                          N_STORED_WORD<=STORED_WORD;
96                                  end if;
97
98                                  ----LOGICA DEL SIGUIENTE ESTADO----
99                                  if (CONTADOR_BIT=8) then
100                                         ESTADO_SIGUIENTE <= STOP;
101                                 else
102                                         ESTADO_SIGUIENTE <= S1;
103                                 end if;
104
105                         when STOP =>
106
107                                  ----SALIDAS----
108                 RX_FLAG<='1';
109
110                                 ----LOGICA DEL SIGUIENTE ESTADO----
111                                  if (READ='1' and INPUT='1') then
112                                          WORD_RECEIVED<= STORED_WORD;
113                                         ESTADO_SIGUIENTE <= IDLE;
114                                         MSG_TRUE<='1';
115                                  elsif (READ='1' and INPUT='0') then
116                                         WORD_RECEIVED<= "00000000";
117                                         ESTADO_SIGUIENTE <= IDLE;
118                                         MSG_TRUE<='0';
119                                  else
120                                          WORD_RECEIVED<= STORED_WORD;
121                                          ESTADO_SIGUIENTE <= STOP;
122                                         MSG_TRUE<='0';
123                                 end if;
124         end case;
125  end process;
126
127
128 process(CLK,REG_ESTADO)
129 variable CONTADOR_RX : INTEGER range 0 to 100;
130 begin
131
132         if(RESET='1') then
133         CONTADOR_RX := 0;
```

```
134         READ<='0';
135         elsif (RISING_EDGE(CLK)) then
136                 if (CONTADOR_RX < 9 and (REG_ESTADO=FB or REG_ESTADO=S1 or REG_ESTADO=STOP)) then
137                         CONTADOR_RX := CONTADOR_RX+1;
138                 else
139                         CONTADOR_RX := 0;
140         end if;
141                 if (CONTADOR_RX=5) then
142                         READ<='1';
143                 else
144                         READ<='0';
145         end if;
146         end if;
147 end process;
148
149
150 process(RX_FLAG)
151 begin
152         BUSY_RX<=RX_FLAG;
153 end process;
154
155 process(READ)
156 begin
157         READTICK<=READ;
158 end process;
159
160 end BEHAVIORAL;
```

B.4. Decodificador

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  use IEEE.NUMERIC_STD;
6
7
8
9  entity DECODIFICADOR_DSPACE is
10         port(
11                 clk : in STD_LOGIC;
12                 reset :in std_logic;
13                 msg_true :in std_logic;
14                 rx_word : in std_logic_vector(7 downto 0);
15                 VC1_ref : out std_logic_vector(1 downto 0);
16                 VC2_ref : out std_logic_vector(1 downto 0);
17                 M_op: out std_logic_vector(1 downto 0);
18                 SM1 : out std_logic_vector(1 downto 0);
19                 SM2 : out std_logic_vector(1 downto 0);
20                 EN_SM1 : out std_logic;
21                 EN_SM2 : out std_logic;
22                 i_ref: out std_logic_vector(11 downto 0)
23                 );
24 end DECODIFICADOR_DSPACE;
25
26
27 architecture Behavioral of DECODIFICADOR_DSPACE is
28 ----Declaración de variables----
29
30 signal  VC1_ref_b: std_logic_vector(1 downto 0);
31 signal  VC2_ref_b: std_logic_vector(1 downto 0);
32 signal  M_op_b: std_logic_vector(1 downto 0);
33 signal        EN_SM1_b: std_logic;
34 signal        EN_SM2_b: std_logic;
35 signal        SM1_b: std_logic_vector(1 downto 0);
```

```
signal        SM2_b: std_logic_vector(1 downto 0);
signal        i_ref_pt1: std_logic_vector(5 downto 0);
signal        i_ref_pt1_b: std_logic_vector(5 downto 0);
signal        i_ref_b: std_logic_vector(11 downto 0);

----LOGICA DE LA COMPONENTE----

---- MSG 1 -----
begin
process (CLK, RESET,rx_word,msg_true)
begin
        if RESET='1' then        --asynchronous RESET active High
                VC1_ref <= "00";
                VC1_ref_b<="00";
        elsif (CLK'event and CLK='1') then  --CLK rising edge
                if ((rx_word(7 downto 6))="00" and msg_true='1') then
                        VC1_ref<=rx_word(1 downto 0);
                        VC1_ref_b<=rx_word(1 downto 0);
                        VC2_ref<=rx_word(3 downto 2);
                        VC2_ref_b<=rx_word(3 downto 2);
                        M_op<=rx_word(6 downto 5);
                        M_op_b<=rx_word(6 downto 5);
                else
                        VC1_ref<=VC1_ref_b;
                        M_op<=M_op_b;
                        VC2_ref<=VC2_ref_b;
           end if;
        end if;
end process;



---- MSG 2 -----
process (CLK, RESET,rx_word,msg_true)
begin
        if RESET='1' then        --asynchronous RESET active High
                SM1 <= "00";
                SM2 <= "00";
                SM1_b <= "00";
                SM2_b <= "00";
                EN_SM1<= '0';
                EN_SM2<= '0';
                EN_SM1_b<= '0';
                EN_SM2_b<= '0';
        elsif (CLK'event and CLK='1') then  --CLK rising edge
                if ((rx_word(7 downto 6))="01" and msg_true='1') then
                        SM1<=rx_word(1 downto 0);
                        SM1_b<=rx_word(1 downto 0);
                        SM2<=rx_word(3 downto 2);
                        SM2_b<=rx_word(3 downto 2);
                        EN_SM1<= rx_word(4);
                        EN_SM2<= rx_word(5);
                        EN_SM1_b<= rx_word(4);
                        EN_SM2_b<= rx_word(5);
                else
                        SM1 <= SM1_b;
                        SM2 <= SM2_b;
                        EN_SM1<= EN_SM1_b;
                        EN_SM2<= EN_SM2_b;
           end if;
        end if;
end process;



---- MSG 3 -----
process (CLK, RESET,rx_word,msg_true)
begin
```

```
104         if RESET='1' then        --asynchronous RESET active High
105                 i_ref_pt1 <= "000000";
106                 i_ref_pt1_b <= "000000";
107 
108         elsif (CLK'event and CLK='1') then  --CLK rising edge
109                 if ((rx_word(7 downto 6))="10" and msg_true='1') then
110                         i_ref_pt1<=rx_word(5 downto 0);
111                         i_ref_pt1_b<=rx_word(5 downto 0);
112 
113                 else
114                         i_ref_pt1<=i_ref_pt1_b;
115            end if;
116         end if;
117 end process;
118 
119 
120 
121 
122 
123 ---- MSG 4 -----
124 process (CLK, RESET,rx_word,i_ref_pt1_b,msg_true)
125 begin
126         if RESET='1' then        --asynchronous RESET active High
127                 i_ref <= "000000000000";
128                 i_ref_b <= "000000000000";
129 
130         elsif (CLK'event and CLK='1') then  --CLK rising edge
131                 if ((rx_word(7 downto 6))="11" and msg_true='1') then
132                         i_ref<=(rx_word(5 downto 0) & i_ref_pt1_b);
133                         i_ref_b<=(rx_word(5 downto 0) & i_ref_pt1_b);
134                 else
135                         i_ref<=i_ref_b;
136            end if;
137         end if;
138 end process;
139 
140 end Behavioral;
```

B.5. Módulo de Disparos

```
1  ----------------------------------------------------------------------------
2  --
3  -- Title       : DISPARADOR
4  -- Design      : MAIN
5  -- Author      : Fernando
6  -- Company     : Federico Santa Maria
7  --
8  ----------------------------------------------------------------------------
9  --
10 -- File        : c:\My_Designs\MEMORIA\MAIN\src\DISPARADOR.vhd
11 -- Generated   : Tue Oct 27 02:25:17 2020
12 -- From        : interface description file
13 -- By          : Itf2Vhdl ver. 1.22
14 --
15 ----------------------------------------------------------------------------
16 --
17 -- Description :
18 --
19 ----------------------------------------------------------------------------
20 
21 ----------------------------------------------------------------------------
22 --
23 -- Title       : DISPARADOR
24 -- Design      : MAIN
25 -- Author      : Fernando
```

```
26  -- Company     : Federico Santa Maria
27  --
28  ------------------------------------------------------------------------------
29  --
30  -- File        : c:\My_Designs\MEMORIA\MAIN\src\DISPARADOR.vhd
31  -- Generated   : Tue Oct 27 02:25:17 2020
32  -- From        : interface description file
33  -- By          : Itf2Vhdl ver. 1.22
34  --
35  ------------------------------------------------------------------------------
36  --
37  -- Description :
38  --
39  ------------------------------------------------------------------------------
40
41  library IEEE;
42  use IEEE.STD_LOGIC_1164.ALL;
43  use IEEE.STD_LOGIC_ARITH.ALL;
44  use IEEE.STD_LOGIC_UNSIGNED.ALL;
45  use IEEE.NUMERIC_STD;
46
47
48
49  entity DISPARADOR is
50          port(
51                  clk : in STD_LOGIC;
52                  reset :in std_logic;
53                  OP_OK :in std_logic;
54                  SM1 : in std_logic_vector(1 downto 0);
55                  SM2 : in std_logic_vector(1 downto 0);
56                  duty_pwm_sp : in integer;
57                  duty_pwm_ss : in integer;
58                  phase_pwm_ss : in integer;
59                  EN_SM1 :in std_logic;
60                  EN_SM2 :in std_logic;
61                  SM1_1 :out std_logic;
62                  SM1_2 :out std_logic;
63                  SM1_3 :out std_logic;
64                  SM1_4 :out std_logic;
65                  SM2_1 :out std_logic;
66                  SM2_2 :out std_logic;
67                  SM2_3 :out std_logic;
68                  SM2_4 :out std_logic;
69                  SP_EN :out std_logic;
70                  SS_EN :out std_logic;
71                  SP_1 :out std_logic;
72                  SP_3 :out std_logic;
73                  SS_1 :out std_logic;
74                  SS_3 :out std_logic
75                  );
76  end DISPARADOR;
77
78
79
80  architecture BEHEAVIORAL of DISPARADOR is
81          signal contador_pwm : integer  range 0 to 36000;
82          signal pwm_sync: std_logic;
83
84  begin
85
86
87
88
89          ----- Generador de disparos para puentes H externos-------
90          process(reset,SM1,EN_SM1)
91          begin
92          if (reset='1') then
93                  SM1_1<='0';
```

```
94                 SM1_2<='0';
95                 SM1_3<='0';
96                 SM1_4<='0';
97
98         elsif (EN_SM1='1') then
99                 SM1_1<=SM1(0);
100                SM1_2<=not(SM1(0));
101                SM1_3<=SM1(1);
102                SM1_4<=not(SM1(1));
103
104        else
105                SM1_1<='0';
106                SM1_2<='0';
107                SM1_3<='0';
108                SM1_4<='0';
109        end if;
110        end process;
111
112        process(reset,SM2,EN_SM2)
113        begin
114        if (reset='1') then
115                SM2_1<='0';
116                SM2_2<='0';
117                SM2_3<='0';
118                SM2_4<='0';
119
120        elsif (EN_SM2='1') then
121                SM2_1<=SM2(0);
122                SM2_2<=not(SM2(0));
123                SM2_3<=SM2(1);
124                SM2_4<=not(SM2(1));
125
126        else
127                SM2_1<='0';
128                SM2_2<='0';
129                SM2_3<='0';
130                SM2_4<='0';
131        end if;
132        end process;
133
134        ----- Modulador PWM SP-------
135        process(clk,reset,duty_pwm_sp,phase_pwm_ss)
136
137        begin
138                if(reset='1' or OP_OK='0') then
139                        contador_pwm <=0;
140                        SP_1<='0';
141                        SP_3<='0';
142                        pwm_sync<='0';
143
144                elsif (rising_edge(clk))
145                then
146                        if (contador_pwm >= (phase_pwm_ss-35)
147                        and contador_pwm <=(phase_pwm_ss+35)) then
148                                pwm_sync<='1';
149                        else
150                                pwm_sync<='0';
151                        end if;
152
153
154                        if ((contador_pwm < 35964) and (duty_pwm_sp>contador_pwm)) then
155                                contador_pwm <= contador_pwm+36;
156                                SP_1<='1';
157                                SP_3<='0';
158
159                        elsif((contador_pwm < 35964) and (duty_pwm_sp<=contador_pwm)) then
160                                contador_pwm<= contador_pwm+36;
161                                SP_1<='0';
```

```
162                             SP_3<='1';
163                         else
164                             contador_pwm <= 0;
165                             SP_1<='0';
166                             SP_3<='0';
167                         end if;
168                 end if;
169         end process;
170
171
172         ----- Modulador PWM SS-------
173         process(clk,reset,duty_pwm_ss,pwm_sync)
174
175                 variable contador_pwm_ss : integer range 0 to 72000;
176         begin
177                 if(reset='1' or OP_OK='0')        then
178                         contador_pwm_ss :=0;
179                         SS_1<='0';
180                         SS_3<='0';
181                 elsif (rising_edge(clk)) then
182                         if ((contador_pwm_ss = 0)
183                         and (duty_pwm_ss>contador_pwm_ss) and (pwm_sync='1')) then
184                                 contador_pwm_ss := contador_pwm_ss+36;
185                                 SS_1<='1';
186                                 SS_3<='0';
187
188                         elsif ((contador_pwm_ss>0 and contador_pwm_ss< 35964)
189                         and (duty_pwm_ss>contador_pwm_ss)) then
190                                 contador_pwm_ss := contador_pwm_ss+36;
191                                 SS_1<='1';
192                                 SS_3<='0';
193
194                         elsif((contador_pwm_ss < 35964) and (duty_pwm_ss<=contador_pwm_ss)) then
195                                 contador_pwm_ss := contador_pwm_ss+36;
196                                 SS_1<='0';
197                                 SS_3<='1';
198
199
200                         else
201                                 contador_pwm_ss := 0;
202                                 SS_1<='0';
203                                 SS_3<='0';
204                         end if;
205                 end if;
206         end process;
207
208         ----- Habilita Enables de Puentes H SP/SS-------
209         process (OP_OK)
210         begin
211                 if (OP_OK='1')        then
212                         SP_EN<='1';
213                         SS_EN<='1';
214                 else
215                         SP_EN<='0';
216                         SS_EN<='0';
217                 end if;
218
219         end process;
220
221 end BEHEAVIORAL;
```

B.6. Unidad de Control Central

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.all;
```

```
3  use IEEE.STD_LOGIC_ARITH.all;
4  use IEEE.STD_LOGIC_UNSIGNED.all;
5  use IEEE.NUMERIC_STD;
6
7  entity UNIDAD_DE_CONTROL_CENTRAL is
8          port(
9                  CLK : in STD_LOGIC;
10                 RESET :in STD_LOGIC;
11                 M_OP :in STD_LOGIC_VECTOR(1 downto 0);
12                 OVER_VOLTAGE :in STD_LOGIC;
13                 OVER_CURRENT :in STD_LOGIC;
14                 OP_OK: out STD_LOGIC;
15                 EM_ON: out STD_LOGIC
16                 );
17 end UNIDAD_DE_CONTROL_CENTRAL;
18
19 architecture BEHAVIORAL of UNIDAD_DE_CONTROL_CENTRAL is
20         type ESTADOS is (IDLE,OPERACION,EMERGENCIA);
21         -- DEFINICIÓN DE LOS ESTADOS DE LA MÁQUINA
22         signal ESTADO_ACTUAL,ESTADO_SIGUIENTE:ESTADOS;
23         -- DECLARACIÓN DEL VECTOR DE ESTADOS
24         signal EMERGENCY_FLG: STD_LOGIC;
25         signal INI_TIME: STD_LOGIC;
26         signal INI_OK: STD_LOGIC;
27 begin
28
29
30
31
32         ---- REGISTRO DE ESTADOS -----
33         process (CLK)
34         begin
35                 if CLK'EVENT and CLK='1' then  --CLK RISING EDGE
36                         if RESET='1' then        --SYNCHRONOUS RESET ACTIVE HIGH
37                                 ESTADO_ACTUAL <= IDLE;
38                         else
39                                 ESTADO_ACTUAL <= ESTADO_SIGUIENTE;
40                         end if;
41                 end if;
42         end process;
43
44
45
46
47
48         ---- LOGICA DE ESTADOS ----
49         process (ESTADO_ACTUAL, M_OP, EMERGENCY_FLG, INI_OK)
50         begin
51                 ESTADO_SIGUIENTE <= ESTADO_ACTUAL;
52                 case (ESTADO_ACTUAL) is
53                         when IDLE =>
54                                 if (INI_OK='1' and EMERGENCY_FLG='0') then
55                                         ESTADO_SIGUIENTE <= OPERACION;
56
57                                 elsif (EMERGENCY_FLG='1')        then
58                                         ESTADO_SIGUIENTE <= EMERGENCIA;
59
60                                 else
61                                         ESTADO_SIGUIENTE <= IDLE;
62                                 end if;
63
64                         when OPERACION =>
65                                 if (EMERGENCY_FLG='1') then
66                                         ESTADO_SIGUIENTE <= EMERGENCIA;
67
68                                 elsif  (M_OP="11" and EMERGENCY_FLG='0')   then
69                                         ESTADO_SIGUIENTE <= IDLE;
70
```

```
71                              else
72                                      ESTADO_SIGUIENTE <= OPERACION;
73                              end if;
74
75                      when EMERGENCIA =>
76                              if (M_OP="11" and EMERGENCY_FLG='0') then
77                                      ESTADO_SIGUIENTE <= IDLE;
78                              else
79                                      ESTADO_SIGUIENTE <= EMERGENCIA;
80                      end if;
81              end case;
82      end process;
83
84
85
86
87      ---- LÓGICA DE LA SALIDA ----
88      process(ESTADO_ACTUAL)
89      begin
90              case(ESTADO_ACTUAL) is
91                      when IDLE => OP_OK <= '0';
92                      EM_ON <= '0';
93                      when OPERACION        =>OP_OK <= '1';
94                              EM_ON <= '0';
95
96                      when EMERGENCIA        => OP_OK <= '0';
97                      EM_ON <= '1';
98              end case;
99      end process;
100
101
102
103     ---- PROCESO DE DETECCIÓN DE FALLA ----
104     process( RESET,CLK,ESTADO_ACTUAL,OVER_VOLTAGE, OVER_CURRENT )
105     begin
106             if (RESET='1')          then
107                     EMERGENCY_FLG<='0';
108             elsif (RISING_EDGE(CLK)) then
109                     if ( (OVER_VOLTAGE='1' or OVER_CURRENT ='1')) then
110                             EMERGENCY_FLG<='1';
111
112                     else
113                             EMERGENCY_FLG<='0';
114                     end if;
115             end if;
116     end process;
117
118
119     ---- PROCESO DEL TEMPORIZADOR DE INICIALIZACIÓN ----
120     process(RESET,CLK,ESTADO_ACTUAL)
121             variable CONTADOR_INI : INTEGER range 0 to 100000;
122     begin
123             if (RESET='1')         then
124                     INI_TIME<='0';
125                     CONTADOR_INI:=0;
126             elsif (RISING_EDGE(CLK)) then
127                     if ((CONTADOR_INI<100000) and (ESTADO_ACTUAL=IDLE)) then
128                             CONTADOR_INI:=CONTADOR_INI+1;
129                             INI_TIME<='0';
130                     elsif ((CONTADOR_INI=100000) and (ESTADO_ACTUAL=IDLE)) then
131                             CONTADOR_INI:=0;
132                             INI_TIME<='1';
133                     else
134                             INI_TIME<='0';
135                             CONTADOR_INI:=0;
136                     end if;
137             end if;
138     end process;
```

```
139
140
141         ---- PROCESO DE INICIALIZACIÓN ----
142         process(INI_TIME,M_OP)
143         begin
144                 if (INI_TIME='1' and (M_OP="00" or M_OP="01" or M_OP="10")) then
145                         INI_OK<='1';
146                 else
147                         INI_OK<='0';
148                 end if;
149         end process;
150
151
152 end BEHAVIORAL;
```

B.7. Interfaz ADC

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.STD_LOGIC_ARITH.all;
4  use IEEE.STD_LOGIC_UNSIGNED.all;
5  use IEEE.NUMERIC_STD;
6
7  entity INTERFAZ_ADC is
8          port(
9                  clk : in STD_LOGIC;
10                 reset :in std_logic;
11                 clk_out_adc :in std_logic;
12                 sd1: in std_logic;
13                 sd2: in std_logic;
14                 sd3: in std_logic;
15                 CNV: out std_logic;
16                 SCK: out std_logic;
17                 VC1: out std_logic_vector(15 downto 0);
18                 VC2: out std_logic_vector(15 downto 0);
19                 Ip: out std_logic_vector(15 downto 0);
20                 adc_ok:         out std_logic
21                 );
22 end INTERFAZ_ADC;
23
24 architecture Behavioral of INTERFAZ_ADC is
25         ---- Declaración de Variables -----
26         --------- CNVH_TIME = (MCLK_freq*50ns)-1 /// 30ns minimo
27         constant CNVH_TIME : integer := 4;
28         --------- TCYC_TIME = (MCLK_freq/FREQ_SAMPLEO)-1 /// 200ns minimo
29         constant TCYC_TIME : integer := 52;
30         -------- TCONV_TIME = (MCLK_freq*150ns)-1 /// 170ns maximo
31         constant TCONV_TIME : integer := 14;
32         --------- TREAD_TIME = CNVH_TIME + TCONV_TIME
33         constant TREAD_TIME : integer := CNVH_TIME + TCONV_TIME;
34         signal reg1 :std_logic;
35         signal reg2 :std_logic;
36         signal SCK_GENERATED :std_logic;
37         signal rise_edge :std_logic;
38         signal fall_edge :std_logic;
39         signal cnv_up: std_logic;
40         signal VC1_b: std_logic_vector(15 downto 0);
41         signal VC2_b: std_logic_vector(15 downto 0);
42         signal ip_b: std_logic_vector(15 downto 0);
43         signal sd1_a: std_logic;
44         signal sd1_sync: std_logic;
45         signal sd2_a: std_logic;
46         signal sd2_sync: std_logic;
47         signal sd3_a: std_logic;
48         signal sd3_sync: std_logic;
```

```
49  --        signal prueba: std_logic;
50
51  begin
52          ----- Detector de Canto: Detecta el canto de clkout para leer la señal del ADC ----
53          process(clk,clk_out_adc,sd1,sd2,sd3)
54          begin
55                  if (reset='1') then
56                          reg1  <= '0';
57                          reg2  <= '0';
58                  elsif rising_edge(clk) then
59                          reg1  <= clk_out_adc;
60                          reg2  <= reg1;
61
62                          sd1_a   <=sd1;
63                          sd1_sync<=sd1_a;
64
65                          sd2_a   <=sd2;
66                          sd2_sync<=sd2_a;
67
68                          sd3_a   <=sd3;
69                          sd3_sync<=sd3_a;
70
71                          rise_edge <= reg1 and (not reg2);
72                          fall_edge <= reg2 and (not reg1);
73                  end if;
74          end process;
75
76
77          ----- CNV: Comanda la señal CNV que incia la conversión -----
78          process(clk,reset,clk_out_adc)
79                  variable contador_cnv: integer range 0 to 100;
80          begin
81                  if (reset='1') then
82                          contador_cnv := 0;
83                          cnv<='0';
84                          cnv_up<='0';
85                  elsif (rising_edge(clk)) then
86                          if (contador_cnv < TCYC_TIME and contador_cnv < CNVH_TIME) then
87                                  contador_cnv := contador_cnv+1;
88                                  cnv<='1' ;
89                          elsif (contador_cnv < TCYC_TIME) then
90                                  contador_cnv := contador_cnv+1;
91                                  cnv<='0' ;
92
93                          else
94                                  contador_cnv := 0;
95                                  cnv<='0';
96                          end if;
97
98                          if (contador_cnv > TREAD_TIME and contador_cnv < (TCYC_TIME-1)) then
99                                  cnv_up<='1' ;
100                         else
101                                 cnv_up<='0' ;
102                         end if;
103                 end if;
104         end process;
105
106
107         ----- SCK: Comanda la señal SCK que es el reloj para la conversión -----
108         process(clk,reset,cnv_up)
109         begin
110
111                 if (reset='1') then
112                         SCK<='0';
113                         SCK_GENERATED <='0';
114                 elsif (rising_edge(clk)) then
115                         if (cnv_up='1') then
116                                 SCK_GENERATED<=not(SCK_GENERATED);
```

```
117                             SCK<=SCK_GENERATED;
118                     else
119                             SCK<='0';
120                     end if;
121             end if;
122     end process;
123
124
125     ----- RECEPCIÓN DE SEÑALES VC1,VC2,IP: RECIBE LAS SEÑALES DEL ADC -----
126     process(clk,reset, rise_edge, sd1_sync, sd2_sync, sd3_sync, VC1_b)
127     variable contador_sdx: integer range 0 to 100;
128     begin
129             if (reset='1') then
130                     contador_sdx := 0;
131                     VC1_b<= "0000000000000000";
132                     VC1<= "0000000000000000";
133                     VC2_b<= "0000000000000000";
134                     VC2<= "0000000000000000";
135                     ip_b<= "0000000000000000";
136                     ip<= "0000000000000000";
137                     adc_ok<='0';
138             elsif (rising_edge(clk)) then
139                     if  (rise_edge='1' and contador_sdx<16) then
140                             VC1_b(15-contador_sdx)<= sd1_sync;
141                             VC1(15-contador_sdx)<= sd1_sync;
142                             VC2_b(15-contador_sdx)<= sd2_sync;
143                             VC2(15-contador_sdx)<= sd2_sync;
144                             ip_b(15-contador_sdx)<= sd3_sync;
145                             ip(15-contador_sdx)<= sd3_sync;
146                             contador_sdx := contador_sdx+1;
147                             adc_ok<='0';
148
149                     elsif (contador_sdx=16)  then
150                             contador_sdx := 0;
151                             VC1<=VC1_b;
152                             VC2<=VC2_b;
153                             ip<=ip_b;
154                             adc_ok<='1';
155
156                     else
157                             VC1<=VC1_b;
158                             VC2<=VC2_b;
159                             ip<=ip_b;
160                             adc_ok<='0';
161                     end if;
162
163             end if;
164     end process;
```

B.8. Declaración de Entradas/Salidas

```
BLOCK RESETPATHS ;
BLOCK ASYNCPATHS ;
LOCATE COMP "CNV" SITE "57" ;
LOCATE COMP "SCK" SITE "49" ;
LOCATE COMP "SM1_1" SITE "75" ;
LOCATE COMP "SM1_2" SITE "74" ;
LOCATE COMP "SM1_3" SITE "71" ;
LOCATE COMP "SM1_4" SITE "70" ;
LOCATE COMP "SM2_1" SITE "63" ;
LOCATE COMP "SM2_2" SITE "62" ;
LOCATE COMP "SM2_3" SITE "65" ;
LOCATE COMP "SM2_4" SITE "64" ;
IOBUF PORT "SM2_4" PULLMODE=DOWN IO_TYPE=LVCMOS25 DRIVE=8 ;
LOCATE COMP "SP_1" SITE "69" ;
LOCATE COMP "SP_3" SITE "68" ;
LOCATE COMP "SP_EN" SITE "61" ;
LOCATE COMP "SS_EN" SITE "60" ;
LOCATE COMP "SS_1" SITE "66" ;
LOCATE COMP "SS_3" SITE "67" ;
LOCATE COMP "sd1" SITE "54" ;
LOCATE COMP "sd2" SITE "53" ;
LOCATE COMP "sd3" SITE "52" ;
LOCATE COMP "clk_out_adc" SITE "51" ;
LOCATE COMP "rst" SITE "19" ;
LOCATE COMP "tx_output" SITE "42" ;
LOCATE COMP "rx_input" SITE "43" ;
LOCATE COMP "op_ok1" SITE "17" ;
LOCATE COMP "em_on1" SITE "18" ;
```