UNIVERSIDAD AUTÓNOMA DE QUERÉTARO

FACULTAD DE INGENIERÍA

"Simplificación de arquitectura para bancos de motores de inducción trifásicos con FPGA SoC, y su implementación para control difuso."

TESIS

QUE PARA OBTENER EL TÍTULO DE

Licenciado en Ingeniería en Automatización

Presenta

Juan Pablo Mateos Paullada

DIRIGIDA POR

José Marcelino Gutiérrez Villalobos

Santiago de Querétaro, Querétaro a 2022

Universidad Autónoma de Querétaro

Facultad de Ingeniería

Licenciatura en Automatización
Electrónica y Sistemas Embebidos

Simplificación de arquitectura para bancos de motores de inducción trifásicos con FPGA SoC, y su implementación para control difuso.

TESIS

Que como parte de los requisitos para obtener el grado de
Licenciado en Ingeniería en Automatización

Presenta:

Juan Pablo Mateos Paullada

Dirigido por:

Dr. José Marcelino Gutiérrez Villalobos

SINODALES

Dr. José Marcelino Gutiérrez Villalobos
Director — Firma

Ing. José Luis Avendaño Juárez
Sinodal — Firma

Dr. Dimas Talavera Vela´zquez
Sinodal — Firma

Dr. Edgar Alejandro Rivas Araiza
Sinodal — Firma

Dr. Manuel Toledano
Director de la Facultad

Ing. José Luis Avendaño Juárez
Coordinador de la Carrera

Centro Universitario
Querétaro, QRO
México.
Noviembre 2022

Esta tesis está dedicada a todas aquellas personas que encuentran un bien mayor retribuyendo al mundo.

Resumen

La presente tesis propone una simplificación en el diseño de los bancos de pruebas para motores de inducción trifásicos que estamos acostumbrados a ver, tanto en libros de control de motores, como en modelos funcionales utilizados en la industria. Dicha simplificación no reduce las capacidades del circuito en cuanto al desempeño, ni en procesamiento. La idea es tener las etapas de control, monitoreo y procesamiento de señales en un sólo circuito integrado, un *FPGA SoC*.

Para justificar la hipótesis, se presenta un modelo práctico del sistema propuesto. Mismo que es explicado a detalle, incluyendo todo el proceso de diseño y las consideraciones teóricas que se tuvieron en cada etapa para poder obtener el resultado deseado.

La propuesta de simplificar la arquitectura, usando como base la tecnología *FPGA* (*Field Programmable Gate Array*) *SoC* (*System on Chip*), de la marca Intel, qué será explicada en el documento. Por un lado, se conecta a una tarjeta de muestreo, qué contiene convertidores analógico digital (*ADC*), junto a una etapa de acoplamiento de señales de los mismos. Por otro lado, se conecta a un inversor trifásico de un kilovatio. El proceso de diseño y la justificación teórica de tanto la tarjeta de muestreo, como el inversor, también serán cubiertos más adelante. Finalmente, se mostrarán los resultados, para que en un futuro se pueda hacer su implementación con lógica difusa. Respondiendo así la hipótesis y objetivos de la tesis.

Tabla de Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

Con la tecnología actual, es posible simplificar la arquitectura de un sistema de pruebas de motores trifásicos, embebiendo el sistema y diseñando modularmente cada etapa.

No hace falta ser un buen observador, para darse cuenta que la cantidad de problemas que surgen día a día, es mucho mayor a la cantidad de soluciones que la humanidad puede ingeniarse para subsistir. Conforme pasa el tiempo, conforme la tecnología avanza, la forma en la que se solucionan los problemas evoluciona, y esta se adapta a los estándares que el aporte colectivo va definiendo. Ciencias relativamente nuevas, como la electrónica, han modificado nuestra forma de vida. Sin embargo, la velocidad de los avances tecnológicos en en dicha área ha superado la capacidad del ser humano para aprender, comprender e implementar dichos avances. Cuando un estudiante que ingresa a un programa profesional de ingeniería electrónica, eléctrica, automatización, o alguna rama similar, se asume que su conocimiento sobre electricidad es casi nulo. Y, el alcanzar a comprender los últimos avances tecnológicos representa un largo camino, lleno de obstáculos, donde una formación sólida de física, electromagnetismo y circuitos eléctricos se vuelve el mejor aliado para la absorción de los conceptos. El estudio de la física, y sobretodo de la electricidad, un fenómeno que es difícil observar en la naturaleza [1], requiere de prácticas que obliguen al estudiante a observar y vivir los fenómenos que nos rodean. Impulsando la curiosidad por la investigación y generando conocimiento empírico sustentado en análisis teóricos.

Un punto importante que se busca probar en esta tesis, es que el camino para llegar a implementar conceptos de última tecnología al final de un grado de licenciatura, requiere de varias horas de dedicación y persistencia al abordar un problema, que eventualmente converge en una solución. Hoy en día, el conocimiento es de libre acceso en varios sitios de internet, en la mayoría de los casos. En este estudio, se busca demostrar que es posible desarrollar circuitos eléctricos con un alto grado de fiabilidad, basados en teorías sencillas y fundamentos. El enfoque, dada la visión de la carrera y la importancia que ha tomado la tecnología en torno a los carros eléctricos en los últimos años, se centrará en el desarrollo un banco de pruebas de motores de inducción trifásicos. Este tipo de bancos son muy comunes en la industria, para simular el funcionamiento del sistema.

Las disciplinas clave del proyecto son la electrónica de potencia (inversor) [2], el control automático (lógica difusa), la instrumentación (sensores), microelectrónica (*Cortex-A9* y *FPGA*) y sistemas operativos (*YOCTO* Linux). El sistema consiste en un motor de inducción trifásico que será impulsado por un inversor trifásico, que será controlado por un *FPGA SoC*, donde se implementarán los controladores y se muestrearán los datos que son obtenidos por un *ADC* a alta

velocidad. Mismos que serán enviados a una computadora para que muestre los datos.

Este proyecto brindará una solución directa, en cuanto a calidad y confiabilidad. Así como una significativa reducción de tamaño a los sistemas ya existentes. El sistema, permitirá a los usuarios tener un conocimiento más amplio de las variables físicas que integran un sistema electrodinámico.

1.1 Motivación

En los últimos años, se observa que las generaciones graduadas en diversas áreas del conocimiento, no tienen la oportunidad de des empeñarse laboralmente en su campo de estudio. Una consecuencia que trae consigo es un menor salario, y una falta de motivación para aportar a la sociedad, ante la gran serie de problemas que surge día con día. Esta tesis surge como un aporte al conocimiento ingenieril, que busca sumar a la tecnología que aquí se menciona en los próximos años y en diversas áreas del conocimiento. Igualmente, el estudio genera una oportunidad para aprender una tecnología capaz de modificar el *"hardware"* en cualquier momento, según su aplicación.

Existen algunos conceptos que erróneos cuando alguien pretende hacer un desarrollo tecnológico. Primero, se asocia el utilizar la última tecnología con un costo elevado. Segundo, muchas veces el acercamiento que se aprende, a lo largo de la carrera, para abordar problemas no es la solución universal. Es difícil pensar fuera de lo convencional. Y tercero, exhortar a los estudiantes de ingeniería y a todo aquél interesado en el documento, a ser conscientes que la barrera del conocimiento es mental, por lo que ningún tema es imposible para aprender e implementar. Mayormente, depende de la dedicación y el esfuerzo que se ponga. Es importante siempre ir más allá de lo que se enseña en clase, porque sólo así se puede comprender la situación tecnológica del mundo y no sólo del entorno estudiantil. Y el nunca perder de vista la importancia de generar aportes a la sociedad y al mundo. A través de este proyecto, se pretende brindar un ejemplo de los resultados que se pueden alcanzar al plantearse una meta complicada.

1.2 Formulación del problema

Cuando se habla del control de un motor trifásico, se habla de un reto un tanto complicado, puesto que el modelo matemático más cercano al comportamiento real, esta descrito por una ecuación diferencial de quinto orden, no lineal [3] y variante en el tiempo. Las técnicas de control buscan simplificar el sistema con aproximaciones matemáticas donde se consideran constantes que simplifican el sistema. A pesar de que matemáticamente las estrategias de control son bastante efectivas, la infraestructura del sistema de control implica un reto en varias áreas de la ingeniería. En la actualidad, los sistemas como el que se pretende construir, son muy grandes y ocupan de mucha circuitería para poder realizar con su labor. En la mayoría de los casos, hay etapas que ocupan una computadora para poder interactuar con el sistema. Y esto, desafortunadamente, en la mayoría de los casos hace que estas herramientas sean muy caras.

En el área de automatización de la facultad de ingeniería no existe un sistema donde los alumnos e investigadores puedan realizar pruebas de control sobre motores de mediana tensión, capaz recibir todos los analíticos al momento de las pruebas, sin miedo a descomponer o dañar el equipo. Contar con un sistema de esta magnitud permitiría a los estudiantes experimentar de una manera más profesional los conceptos vistos en materias como: electrónica de potencia, máquinas eléctricas

e instrumentación. La gran ventaja de este sistema es que simula una situación, como si el estudiante estuviera en un entrenamiento para una compañía, en un entorno industrial, pero sin perder la protección física y didáctica que ofrece un laboratorio. Además que se podrán observar los analíticos que muchas veces se ignoran al momento de una experimentación. De modo, que toda esta información que se demuestra en la teoría podrá compararse con la práctica.

El avance de la tecnología no sólo se ve reflejado cuando se descubre algo nuevo, si no también cuando se mejora lo que ya existe. Eso es lo que se pretende lograr con esta tesis, utilizar tecnología nueva, que no se ha popularizado, en aplicaciones donde puede tener un muy buen desempeño y traer consigo muchas otras ventajas al sistema como bajo consumo de energía, menor tamaño y mayor velocidad de procesamiento. Dadas las necesidades que existen en la industria para realizar pruebas sobre máquinas eléctricas, la idea es construir una estación didáctica que integré todas las herramientas en una sola estación, para que el usuario pueda experimentar los conocimientos de electrónica de potencia, máquinas eléctricas y control automático; para mover motores y observar las variables físicas en tiempo real. Una vez llegado el momento de ejecutar alguna prueba o práctica de laboratorio, se dará por hecho que el sistema funciona de forma correcta y que los valores presentados por la interfaz son correctos. Todo esto garantizando la seguridad de el usuario, ya que vale la pena recalcar que cuando se habla de electrónica de potencia, se debe tener extrema cautela, dado que la vida puede ponerse riesgo si no se toman las medidas adecuadas de precaución. Las tensiones son elevadas y las corrientes qué se utilizan pueden llegar a poner en riesgo la integridad de una persona, con el menor descuido.

1.3 Hipótesis y Objetivos

1.3.1 Hipótesis

Es posible construir una estación de pruebas y monitoreo que sea capaz de operar y visualizar variables de motores de corriente alterna trifásicos, basada en un sistema embebido de alta velocidad con una interfaz en tiempo real.

1.3.2 Objetivo General

El objetivo principal es diseñar y construir una estación didáctica que integre a través de una interfaz en tiempo real, elementos de control de motores de corriente alterna trifásica, elementos de protección y aislamiento, y sensores de medición de voltaje, corriente, velocidad y torque para motores eléctricos.

1.3.3 Objetivo Particular

- Diseñar una tarjeta de muestreo de alta velocidad para instrumentar los motores, a través de sensores de voltaje, corriente, velocidad.
- Integrar el procesamiento de datos la etapa de control e instrumentación, con una interfaz gráfica utilizando sólo un circuito integrado para el procesamiento, un *FPGA SoC*.
- Programar una interfaz gráfica capaz de mostrar al usuario en tiempo real los valores de las variables que se están monitoreando.

- Diseñar y construir un inversor trifásico, en una tarjeta circuito impreso, capaz de suministrar la alimentación a un motor de inducción trifásico.

1.4 Estructura de la Tesis

La estructura de esta tesis, está basada en la distribución de información que contienen algunos libros técnicos, que he tenido la fortuna de leer y que han sido bastante explicativos en cuanto a la presentación de los antecedentes que se ocupan para comprender posteriormente temas más complejos. Muchas veces, si no se tiene la experiencia, ni el conocimiento previo para comprender ciertos conceptos, es muy pesado para el lector querer adquirir nuevos conocimientos y entender la aplicación de los mismos, simultáneamente. Es por eso que el capítulo 2 está dedicado a presentar todos los antecedentes teóricos de una forma resumida, para que el lector tenga la capacidad de concentrarse en comprender el objetivo principal de la tesis, y en caso de tener alguna duda pueda rápidamente volver y retomar el hilo. Los conceptos técnicos introducidos en este capítulo están presentados de una manera digerida, con el objetivo de aterrizar estos conceptos con la metodología de la tesis. Se le invita al lector, en el caso de tener alguna duda, consultar la bibliografía que esta citada a lo largo de todo el documento.

El capítulo 3 presenta la metodología. Se plantean las especificaciones y requisitos con los cuales debe cumplir el sistema a diseñar. Así como al acercamiento que se tuvo a un nivel parcial, incluidas las iteraciones fallidas y el aprendizaje de las mismas, para así llegar al proceso definitivo que cumple con todos los requerimientos ya mencionados.

El capítulo 4, se presentan los resultados de todo el proceso de experimentación. Se presentan de forma gráfica los circuitos y tarjetas que se diseñaron, se muestra la efectividad de dichas tablillas, así como el desempeño de los algoritmos desarrollados para esta aplicación. Se muestran las gráficas con los voltajes y corrientes del sistema al arrancar el motor, variando la carga inicial, así como las limitantes de diseño y procesamiento del sistema.

Finalmente, se concluye con base a los resultados obtenidos, se comparten las lecciones aprendidas y las áreas de mejora en cada etapa del sistema. Probando la hipótesis establecida en éste documento.

Capítulo 2

Antecedentes Teóricos

El estudio de la electrónica no es una tarea sencilla, y es complicado encontrar un punto de partida para empezar a entrelazar los diferentes conceptos. Ciertamente, cuando se lee un libro de la materia, una formación previa de álgebra y cálculo ayudan a comprender mejor los conceptos, cuando se presenta una lluvia de ecuaciones que muchas veces termina por confundir al lector. En este documento, los fundamentos y los conceptos de esta lección se explican de forma resumida y simplificada, y se invita al lector hacer referencia a la bibliografía en caso de tener la inquietud de investigar más a detalle los conceptos presentados.

2.1 Fundamentos Físicos

2.1.1 Corriente

La corriente eléctrica se refiere a una magnitud que cuantifica el número de electrones (carga) que pasan a través de un camino, en un determinado tiempo. Muchas veces al momento de la aplicación práctica, se simplifica o se obvia el concepto de corriente. Sin embargo, al momento de experimentar desde un punto de vista de electromagnetismo, y buscando comprender con mayor detalle el fenómeno, es importante apegarse a la definición de la magnitud.

$$i = \frac{dq}{dt} \tag{2.1}$$

La definición matemática utilizada a lo largo de este documento esta expresada en la ecuación 2.1. Donde se describe como una razón de cambio de una carga con respecto al tiempo. La unidad de la corriente es el amperio, que es igual a la carga en culombios, entre un tiempo en segundos.

2.1.2 Voltaje

Dado que las cargas en un conductor se mueven de forma aleatoria, se ocupa un trabajo o energía para mover las cargas a través de un conductor. Esta energía empleada para desplazar las cargas entre dos puntos de le llama voltaje. Su unidad es el voltio, que es igual a una energía en julios usada para desplazar una carga en culombios.

$$v = \frac{dw}{dq} \tag{2.2}$$

2.1.3 Potencia

La potencia es una unidad derivada del voltaje y la corriente. Y ayuda a cuantificar cuanta energía se transfiere al circuito para su consumo. En casos prácticos, si la unidad de potencia es positiva, significa que el sistema de referencia esta disipando energía. En el caso contrario, si la unidad es negativa, se refiere que el sistema de referencia esta entregando energía.

$$p = \frac{dw}{dt} = v \cdot i \tag{2.3}$$

La potencia se expresa en vatios, que es igual a la energía consumida con respecto al tiempo.

2.1.4 Resistencia

La resistencia eléctrica es una propiedad de los materiales que describe una oposición al flujo de corriente. De modo que un material con resistencia muy baja, se considera un conductor, puesto que la oposición al flujo de corriente es imperceptible. Los metales son buenos ejemplos de conductores. Sin embargo, es importante considerar que la resistencia es una propiedad extensiva, por lo que un cable de dos metros, tendrá el doble de resistencia de un cable de un metro. En el caso que la resistencia eléctrica de un material es alta, se le considera un aislante, puesto que la oposición a la corriente es muy grande.

Una resistencia es también un componente básico en la electrónica, que consta de una resistencia eléctrica definida. Su función es disipar la energía que se pierde con la fricción que genera la corriente que fluye a través del componente. Cada componente cuenta con una tolerancia, un rango dentro del que se encuentra el valor de resistencia real, con respecto al señalado; una potencia de disipación máxima; un tamaño de empaquetado, mismo que se elige según la aplicación y características.

2.1.5 Ley de Ohm

Existe una relación entre voltaje, corriente y resistencia. Que establece que un voltaje a través de una resistencia, es directamente proporcional a la corriente que fluye a través de ella.

$$v = R \cdot i \tag{2.4}$$

2.1.6 Conductancia

La conductancia es el inverso de la resistencia. Físicamente, la conductancia es una forma de medir que tan bien un material conduce corriente eléctrica. La unidad de la conductancia son los siemens.

$$G = \frac{1}{R} \tag{2.5}$$

2.2 Electrónica

2.2.1 Fundamentos

Componentes Pasivos

Los componentes pasivos son aquellos que no tienen la capacidad de introducir energía al circuito, y no necesitan de alguna fuente externa para funcionar. Además de la resistencia, existe el inductor y el capacitor, que se describirán a continuación.

Inductor Un inductor, en términos prácticos, se forma enrollando un conductor en algún metal, o simplemente aire. Que al aplicar corriente a un inductor, se genera un campo electromagnético que se acumula en y se opone al los cambios en la corriente. A esta oposición en los cambios de corriente se le llama inductancia, y su unidad son los henrios.

$$i(t) = i(0) + \frac{1}{L}\int_0^t [v(t)dt] \tag{2.6}$$

Capacitor Un capacitor, en términos prácticos, se forma poniendo dos placas de conductor de forma paralela, donde entre las placas puede ir algún dieléctrico, o simplemente aire. Al aplicar un voltaje sobre sus terminales, estas se polarizan y acumulan una carga entre el par de placas paralelas. Esta carga va a generar oposiciones a los cambios de voltaje, esta característica de acumular carga se le conoce como capacitancia, y su unidad son los faradios.

$$v(t) = v(0) + \frac{1}{C}\int_0^t [i(t)dt] \tag{2.7}$$

Circuito *RCL*

Cuando una resistencia, un capacitor y un inductor se encuentran en el mismo circuito se le conoce como un circuito *RCL*. Este circuito tiene dos variantes principales, donde los componentes están conectados en serie, o en paralelo. Mismos que sientan las bases para modelar sistemas complejos como lo es un modelo equivalente de un motor de inducción trifásico, y dan una noción como se comportan.

2.2.2 Electrónica Analógica

Amplificadores Operacionales

El transistor es un componente que puede ser utilizado en tres regiones de operación, siendo la región activa la más importante en su momento, porque permitía amplificar una señal. Un amplificador operacional es un dispositivo con una ganancia de salida elevada, que a diferencia del transistor, tiene mayor estabilidad y robustez al momento de amplificar. El descubrimiento de este dispositivo abrió pasó a todo un mundo dentro de la electrónica analógica. Ya que con sus configuraciones definidas por componentes externos, es posible realizar operaciones matemáticas básicas e incluso derivadas e integrales. Con el paso del tiempo, se han hecho investigaciones y mejoras a este dispositivo, para garantizar que la señal de salida no se vea contaminada por algún ruido dentro del proceso de amplificación.

Un amplificador operacional cuenta con dos entradas de alimentación, donde según sus características eléctricas, se deberá aplicar un voltaje de una fuente. En la mayoría de los casos, se deberá contar con una fuente de voltaje positivo y una fuente de voltaje negativo. El amplificador operacional cuenta con dos entradas (inversora y no inversora) de alta impedancia, en las cuales se aplica la señal de entrada que se pretende amplificar. Es importante mencionar que estas dos entradas, idealmente, deberán tener 0V entre ellas. Y una señal de salida.

Un factor muy importante que debemos considerar al momento de realizar diseños con amplificadores operacionales es que no son dispositivos ideales. Y existen limitaciones físicas a la entrada y a la salida, que se deben considerar al momento de realizar los análisis y esquemáticos, para

poder tener un mejor desempeño con los dispositivos, mismas que cobrarán sentido en el capítulo de compatibilidad electromagnética.. Algunas de estas limitaciones se presentan a continuación.

Rechazo al Modo Común Dentro de la hoja de especificación de cualquier amplificador operacional, existe una característica eléctrica llamada proporción de rechazo al modo común que está dada en decibelios (db), por lo que este valor varía entre modelo y fabricante. A lo que se refiere el rechazo en modo común, esa la capacidad del dispositivo para minimizar el efecto en la amplificación de salida, cuando exista una diferencia de voltaje entre las dos terminales de entrada. Es importante mencionar que en un dispositivo real la diferencia de voltaje entre las dos terminales no es 0V. Sin embargo, los fabricantes han mitigado este fenómeno con métodos de aislamiento, de balanceo, o alguna combinación entre ambos. Es importante considerar que el rechazo al modo común decrementa como una función de la frecuencia, dada la influencia de capacitancias parásitas entre las terminales entrada.

Compensación en Frecuencia Al momento de amplificar una señal, cada etapa de amplificación que genera un retraso de fase. Donde la ganancia de la amplificación se ve limitada por la frecuencia, ya que cuando el retraso en la fase supera los 180° , la ganancia en lazo abierto debe ser menor a uno. En caso de no tener una ganancia menor a la unidad, la retroalimentación del lazo de control genera un sistema auto oscilante. La compensación en frecuencia es necesaria para los amplificadores con más de una etapa. Lo recomendado es controlar el amortiguamiento de la señal, para esto la compensación puede ser interna o externa. Normalmente, la solución más directa es colocar un filtro RC que establezca una frecuencia dominante y a partir de dicha frecuencia, atenuar las señales.

Slew Rate Durante la práctica, el comportamiento de los amplificadores operacionales no es ideal. Los dispositivos se ven limitados en términos de corriente y voltaje por la naturaleza del semiconductor, el proceso de manufactura y el diseño de los componentes. Las limitantes de corriente restringen la velocidad a la que puede cambiar el voltaje de salida en un amplificador operacional. Esta velocidad se le conoce como *slew rate*. Que está representado por la siguiente ecuación:

$$S_r = \left(\frac{dV_{out}}{dt}\right)_{max} \tag{2.8}$$

Es importante considerar que el *slew rate* está limitado por la velocidad de la etapa de entrada, que está definida por la compensación en frecuencia. Una buena regla de diseño, encontrar la frecuencia de salida cuya ganancia es igual a $0db$. Y definir la compensación en frecuencia de la entrada del amplificador a la mitad de esta frecuencia máxima de salida. De esta forma, el sistema quedará definido por razón de cambio de voltaje en la entrada. Y se podrá garantizar que la salida cumplirá con los requerimientos de velocidad.

Configuración - Diferencial La configuración diferencial es una de las aplicaciones más comunes del amplificador operacional. Dicho arreglo se muestra en la figura 2.1 da como salida la diferencia de voltaje entre las dos entradas, misma que puede ser amplificada por un escalar, según las resistencias externas que se coloquen. El voltaje de salida esta descrito por la ecuación 2.9.

$$V_{out} = -V_1\left(\frac{R_3}{R_1}\right) + V_2\left(\frac{R_4}{R_2+R_4}\right)\left(\frac{R_1+R_3}{R_1}\right) \tag{2.9}$$

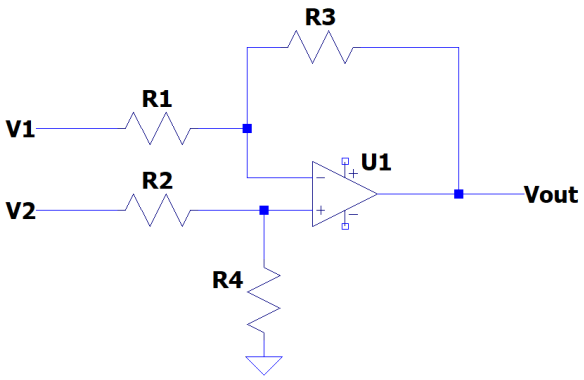


Figura 2.1: Amplificador operacional en configuracion diferencial.

En el caso que las resistencias R1 y R2 sean iguales, y que R3 y R4 tengan el mismo valor. La ecuación 2.9, puede simplificarse a la ecuación 2.10.

$$V_{out} = \frac{R_3}{R_1}(V_2 - V_1) \tag{2.10}$$

Convertidor Analógico-Digital (ADC)

Dados los avances tecnológicos orientados hacia la electrónica digital, fue necesario realizar un método para poder interpretar señales analógicas en cifras binarias, para posteriormente ser procesados en un sistema digital. Los convertidores analógico-digital poseen una frecuencia de muestreo f_s, mismo que debemos considerar con el objetivo de evitar el fenómeno de *aliasing*. El *aliasing* fenómeno inherente de la cuantización de una señal. Y se presenta cuando la frecuencia de muestreo no es el doble de la frecuencia de la señal a medir. Puesto que la medición se realiza en puntos. El no tener una frecuencia de muestreo de la magnitud mencionada, genera una mala interpretación de la señal adquirida. Dicha interpretación puede estar reflejada como una variación en la frecuencia de la misma. De modo que frecuencia de la señal a muestrear, debe ser por lo menos la mitad de la frecuencia de muestreo $f_s/2$.

Los convertidores analógico-digital poseen una resolución y un voltaje de referencia. La precisión del dispositivo está dada al dividir el voltaje de referencia entre la resolución.

$$Precisión = \frac{V_{ref}}{Resolución - 1} \tag{2.11}$$

Posteriormente, si la señal medida en la entrada es menor o igual al voltaje de referencia, se le asignará un valor proporcional en bits, al número de intervalo que le corresponde. Por ejemplo, si la referencia de mí convertidor es de 1V, y la resolución es de 4 bits (16 unidades) y se le induce una señal de 0.5V en la entrada. Entonces, la precisión de mi convertidor será de $1V/15bit = 0.0667\frac{V}{bit}$, entonces haciendo la relación para encontrar el intervalo correspondiente a mi señal de entrada sera $0.5V/0.0667\frac{V}{bit} = 7.5$ que se redondea a $7bit$. En el caso anterior, existe un error muy grande por redondear las cantidades a números enteros, a causa de la incapacidad del dispositivo de poder distinguir los valores entre los intervalos establecidos. A este error se le conoce como error de

cuantización, y es inherente al proceso de digitalización. Evidentemente, la manera más directa de reducirlo, es incrementando la resolución del dispositivo. Al momento de utilizar en la aplicación un convertidor analógico-digital, es importante tomar en cuenta que cada conversión requiere cierto tiempo para realizar el proceso, estabilizarse, y tener el valor de la medición. A este tiempo se le conoce como tiempo de conversión, mismo que varía según el método de conversión y del dispositivo en sí.

2.2.3 Electrónica de Potencia

Fuentes de Alimentación

La electrónica de potencia estudia el procesamiento y el control de la energía, en forma de voltajes y corrientes, que se suministra de la manera más óptima a los sistemas eléctricos, según sus requisitos de funcionamiento. Los voltajes que se controlan pueden ser tanto corriente directa, como corriente alterna. Dentro del estudio de la electrónica de potencia se encuentra el diseño de convertidores, que son etapas sencillas de conversión energía eléctrica [4]. Existen cuatro tipos de convertidores:

- Convertidores de CA a CD (rectificadores).
- Convertidores de CD a CD.
- Convertidores de CD a CA (inversores).
- Convertidores de CA a CA.

Cuando hablamos de convertidores de voltaje de CD a CD, una de las formas de categorizarlos es en reguladores lineales y reguladores conmutados. La diferencia entre este tipo de fuentes radica en el método utilizado para realizar la conversión de la energía. Para cada tipo de convertidor existen diferentes métodos para realizar dicha conversión de energía eléctrica. Los métodos utilizados para esta tesis serán descritos a continuación.

Reguladores Lineales Las fuentes lineales regulan el voltaje de salida disipando el voltaje con un componente en serie. Este método es poco eficiente, puesto que disipa mucha potencia en forma de calor, por lo que sólo es recomendado para aplicaciones de $10W$ o menos. Un requisito fundamental para este tipo de reguladores es que el voltaje de salida debe ser menor al voltaje de entrada. Existen dos tipos de reguladores lineales, los reguladores *shunt* y los reguladores *series-pass*. Los reguladores *shunt* se ponen en paralelo con la carga y el controla la corriente de la carga para mantener el voltaje deseado. Un ejemplo es un regulador basado en un diodo *Zener*. Los reguladores *series-pass* son más eficientes puesto que usan un semiconductor como el elemento que regula la tensión de salida. Tienen un lazo de control, donde el error de voltaje del lazo controla la conductividad del semiconductor para mantener el valor deseado de voltaje en la salida. La velocidad de respuesta va a ser dependiente del error en la compensación del lazo de retroalimentación, este parámetro depende del diseño en la fabricación del componente, y también definirá su respuesta en alta frecuencia, el ancho de banda y el tiempo para responder a una respuesta transitoria. Los reguladores lineales son utilizados para suministrar voltaje a una carga que sea sensible al ruido, cuando se tiene una fuente de alimentación con ruido en la entrada. Son bastante populares por ser muy fáciles de usar, ya que ocupan pocos componentes externos y son muy accesibles en cuanto a su costo.

Los parámetros que se deben considerar al momento de elegir un regulador lineal son:

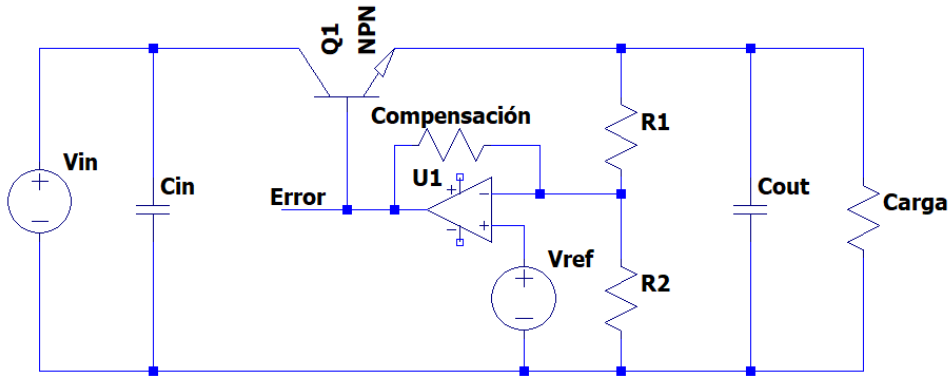


Figura 2.2: Lazo de control de un regulador lineal de voltaje.

- Voltaje de *Dropout*. Es la diferencia de voltaje entre la entrada y la salida, tal que si el voltaje de entrada se reduce, el voltaje de salida regulado se verá afectado. Con el paso de los años este valor se ha ido reduciendo cada vez más, sobretodo desde la implementación de nuevos semiconductores. El rango de este valor suele ser de entre 700mV y 30mV.
- Regulación de la carga (LoR). Indica el desempeño del elemento pasivo y la ganancia de CD en lazo abierto. A mayor ganancia en lazo abierto, mejor regulación en la carga.

$$LoR = \frac{\Delta V_{out}}{I_{out}} = \frac{V_{out@no-load} - V_{out@full-load}}{0 - I_{out@full-load}} \tag{2.12}$$

- Regulación en la línea (LiR). Define la capacidad del regulador para mantener el voltaje de salida cuando se varía el voltaje de entrada. Es dependiente del desempeño del elemento pasivo y la ganancia de CD en lazo abierto.

$$LiR = \frac{\Delta V_{out}}{\Delta V_{in}} = \frac{V_{out@V_{in}max} - V_{out@V_{in}min}}{V_{in}max - V_{in}min} \tag{2.13}$$

- Corriente inactiva (*quiescent*). Es la diferencia entre la corriente de entrada y la corriente de salida, a menor valor de corriente inactiva, mayor será la eficiencia del regulador. La corriente inactiva está determinada por los elementos en serie, la topología, la temperatura, entre otros factores.

$$i_q = i_i - i_o \tag{2.14}$$

- Corriente de espera (*standby*). Es la corriente consumida Por el regulador cuando el voltaje de salida está desactivado.
- Eficiencia. Para tener una alta eficiencia es necesario tener corriente inactiva y el voltaje de *dropout* mínimos.

$$\eta = \frac{i_o * v_o}{(i_o + i_q)v_i)} * 100 \tag{2.15}$$

- El rechazo al rizo. Es la capacidad de prevenir las fluctuaciones en el voltaje de salida causadas por la variación en el voltaje de entrada. Un capacitor con baja resistencia equivalente en serie y alto capacitancia, la salida del circuito puede ayudar a mejorar su desempeño [5].

Regulador *Charge Pump* El regulador *charge pump* es un regulador conmutado qué se basa en el principio de cargar y descargar capacitores. Este regulador es utilizado aplicaciones de baja corriente. Tiene la ventaja de que no ocupan un inductor, por lo que la implementación se puede lograr un espacio reducido. Además, su eficiencia es superior a la de un regulador lineal. Y el voltaje de entrada puede ser mayor, o menor al voltaje de salida. Sin embargo, voltaje de salida puede ser ruidoso y puede tener un voltaje de rizo elevado. Este regulador también tiene la capacidad de

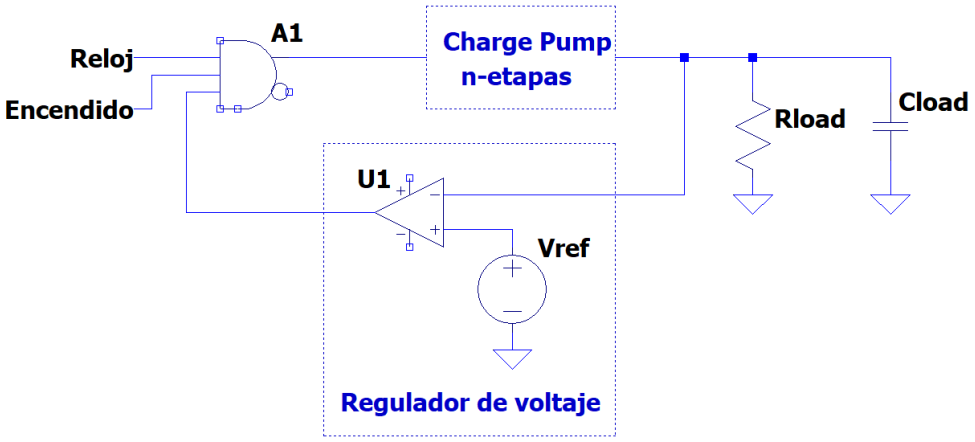


Figura 2.3: Lazo de control en regulador *charge pump*

multiplicar el voltaje, así como de negar el voltaje de salida. La magnitud en la que se multiplica depende del número de etapas que se colocan en serie. Las etapas de muestran en la figura 2.4.

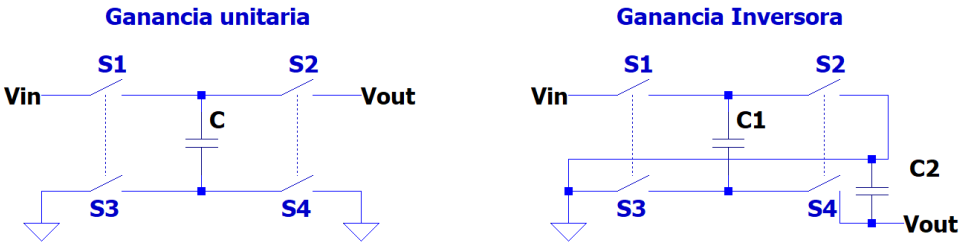


Figura 2.4: Etapas del regulador *charge pump*

Hoy en día las soluciones que los proveedores de componentes ofrecen, suelen tener un regulador *boost*, para suministrar voltaje a la entrada del regulador *charge pump*.

Inversor Los inversores son convertidores de CD a CA. Su importancia surgió a partir de la invención de componentes como el *IGBT* y *MOSFET*, a partir de ese momento y se empezaron a inventar diversos métodos de control, ya que la conmutación de los dispositivos podía ser regulada a través de pulsos de bajo voltaje [2].

Las aplicaciones principales del inversor son:

- Variadores de voltaje / variadores de frecuencia qué son usados como controladores de velocidad para motores de corriente alterna [6].
- Fuentes de poder de corriente alterna.
- Filtros pasivos y filtros activos.
- Compensadores de voltaje.

Las técnicas de conversión de CD a CA se divide en modulación de ancho de pulso (PWM) y modulación multinivel. Para los alcances de esta tesis, el enfoque será en la modulación de ancho de pulso. Para el método de modulación de ancho de pulso se deberá contar con un voltaje de entrada constante. También, existen diferentes tres tipos principales de inversores donde dependiendo del tipo, las técnicas de modulación varían, puede ser como fuente de voltaje (VSI), como fuente de corriente y como fuente de impedancia. Con el objetivo de controlar motores, se seguirá el acercamiento del inversor fuente de voltaje [7].

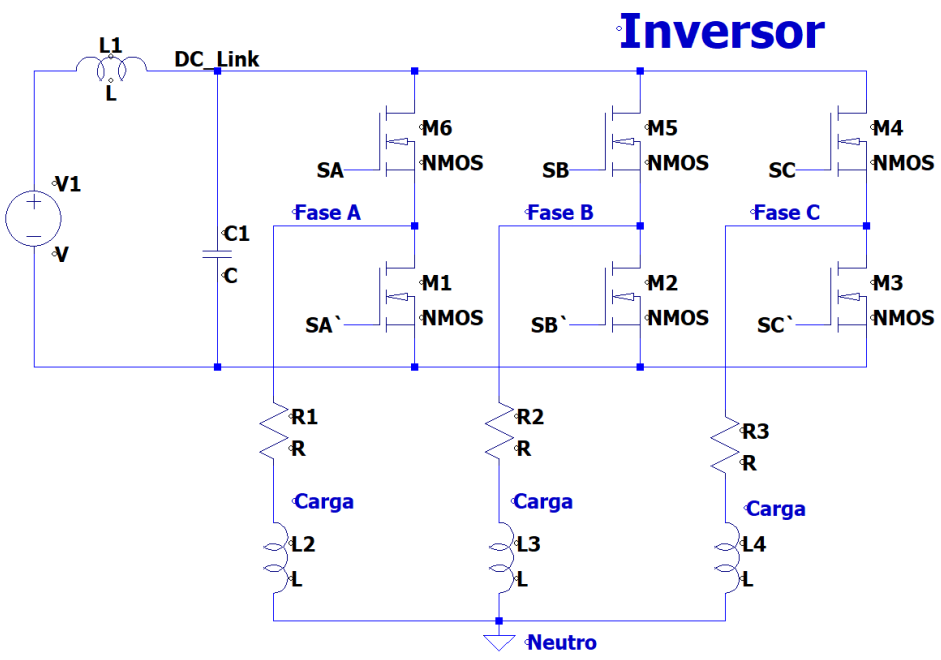


Figura 2.5: Circuito inversor de tres fases.

Las voltaje de salida objetivo al diseñar un VSI suele estar entre $1Hz$ y $400Hz$, donde el pulso con modulación (PWM) de la señal portadora es recomendado a una frecuencia de entre $10Hz$ a $20kHz$, dejando un espacio de tiempo muerto de entre $20ns$ y $100ns$ entre cada cambio de etapa. Para el diseño de un inversor trifásico se deben desfasar las señales $120°$ una de otra. Para lograr hacer las conmutaciones sin crear un corto circuito, es recomendado seguir el diagrama de tiempos mostrado a continuación [8].

Numero en la Secuencia	*MOSFET* activo		Fases		
	Puente Alto	Puente Bajo	A	B	C
1	C	B	No	-	+
2	A	B	+	-	No
3	A	C	+	No	-
4	B	C	No	+	-
5	B	A	-	+	No
6	C	A	-	No	+

Tabla 2.1: Diagrama de tiempos *VSI*

2.2.4 Electrónica Digital

A diferencia de la electrónica analógica, la electrónica digital es ubicua, y no es susceptible a variaciones porque se limita a los niveles de voltaje considerados como estados alto (1) y bajo (0). A partir de ese principio binario se desarrolla toda una teoría de álgebra booleana, que abstrae en ceros y unos las matemáticas conocidas y fundamenta toda la revolución tecnológica de los últimos años [9].

Compuertas Lógicas

Para poder realizar operaciones matemáticas en términos booleanos, es necesario un arreglo electrónico que sea capaz de recibir entradas para emitir un resultado de salida. Las compuertas lógicas son el elemento más básico en la lógica combinacional. Existen tres compuertas lógicas fundamentales que son AND, OR y NOT. A partir de ellas se derivan otras como la NAND, NOT, XOR y XNOR. Cada una de ellas tiene su tabla de verdad que describe la salida esperada a partir de las combinaciones posibles en sus entradas [10].

Compuerta OR La compuerta OR es una abstracción de una suma. Donde si se tienen dos entradas, A y B. La salida Y será igual a $(A||B)$ que es lo mismo que $(A+B)$. Su tabla de verdad se muestra a continuación en la tabla 2.2.

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Tabla 2.2: Tabla de verdad compuerta OR.

Compuerta AND La compuerta AND es una abstracción de una multiplicación. Donde si se tienen dos entradas, A y B. La salida Y será igual a $(A\&B)$ que es lo mismo que $A*B$. Su tabla de verdad se muestra a continuación en la tabla 2.3.

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 2.3: Tabla de verdad compuerta AND.

Compuerta NOT La compuerta NOT es una abstracción de una negación. Donde si se tiene una entrada A, a salida será igual a $\bar{A}$ y viceversa.

Memorias

Al incrementar la complejidad de los sistemas digitales, surge la necesidad de acumular valores pasados. Para que exista un valor pasado se debe generar una referencia en cuanto a tiempo, misma que fue representada por las señales de reloj, que son patrones de pulsos que iteran entre el estado alto y bajo, y se repiten de manera síncrona. La capacidad de almacenar datos dio paso a diferentes estrategias y tecnologías, todas basadas en una lógica secuencial, mismas que se muestran a continuación.

Flip Flop Son arreglos de compuertas lógicas que permiten almacenar un valor binario, mismo que es conocido como *bit*, cuando se coloca un arreglo de 8 elementos de estos componentes, la unidad es conocida como *byte*. Existen diferentes tipos de *flip flops*, pero para los alcances de esta tesis nos limitaremos explicando el funcionamiento del tipo *Set - Reset* (*SR*). Al igual que las compuertas lógicas, poseen una tabla de verdad. En el caso de este tipo en específico, este *flip flop* no cuenta con una señal de reloj (asíncrono), y es conocido como *latch*. La tabla de verdad del *latch SR*, se muestra en la tabla 2.4.

Estado	***S***	***R***	***Q***	***Evento***
Set	1	0	1	$Q \rightarrow 1$
	1	1	1	Sin cambio
Reset	0	1	0	$Q \rightarrow 0$
	1	1	0	Sin cambio
No valido	0	0	1	Sin cambio

Tabla 2.4: Tabla de verdad *latch SR*.

Flash Las memorias *Flash* se caracterizan por ser no volátiles, es decir que una vez que se desconectan de su fuente de alimentación tienen la capacidad de retener sus datos por varios años. Una memoria no volátil guarda el estado definido hasta que se fuerza un cambio de estado, normalmente a través de un pulso eléctrico. Por un lado están las memorias con arquitectura NOR y por otro lado las de arquitectura NAND. Actualmente, las memorias *Flash* se utilizan en la mayoría de los aparatos electrónicos dado su bajo costo, velocidad de escritura y capacidad de almacenamiento. Desde guardar fotos en una memoria *USB* (*Universal Serial Bus*), un disco duro

de estado sólido, el chip que guarda el *BIOS* (*Basic Input/Output System*) de la computadora; todos son ejemplos de aplicaciones de memorias de este tipo.

Memoria *DDR RAM* Las memorias *RAM (Random Access Memory)* han tomado una importancia fundamental en el era tecnológica que se vive. Dadas las capacidades de procesamiento que incrementan año con año, definidas por la ley de Moore, las necesidades en cuanto a velocidad de escritura y lectura en memorias también aumentan, ya que en caso de no hacerlo dicho procesamiento no serviría de nada. Este tipo de memorias se caracteriza por ser volátiles, es decir, que al momento de cortar el voltaje de alimentación de entrada, perderán todo su contenido. Las memorias *DDR (Double Data Rate)* son el resultado de años de investigación; teniendo como antecedente las memorias *DRAM*, que buscaron dar una solución asíncrona que fue rebasada por la velocidad de los procesadores; las memorias *SDRAM*, que regresaron a una funcionalidad síncrona y eran escritas o leídas en un solo ciclo de reloj. La tecnología *DDR* es capaz de realizar lecturas o escrituras de forma síncrona dos veces por ciclo de reloj, primero en el flanco de subida y después en el flanco de bajada, de ahí su importancia en aplicaciones de cómputo hoy en día.

Las memorias *DDR* a lo largo del tiempo han sido categorizadas por sus capacidades de operación. Actualmente, la memoria de uso común es la *DDR4*, siendo la 4ta generación de memorias. Sin embargo, en 2021 se introdujo al mercado de la *DDR5*, que duplica la velocidad de la generación anterior y para 2024 la *DDR6*.

Generación	Voltaje de Operación	Velocidad (MHz)
DDR	2.5 V	100 – 200
DDR2	1.55 V	200 – 400
DDR3	1.35 V	400 – 1066
DDR4	1.2V	1066 – 3200

Tabla 2.5: Evoluci´on en las capacidades de las memorias *DDR*

FPGA

La invención de los *FPGA (Field Programable Gate Array)* significó avances bastante significativos al área de electrónica digital. Aún así, hasta hace algunos años esta tecnología era más utilizada en el área académica, que en desarrollos para productos finales. Se le conoce como *FPGA* porque esta constituido por un arreglo de bloques lógicos, y programable en campo porque puede ser programado fuera de la fábrica donde se construyó el componente (esto pensando que cuando se inventó los circuitos eran programados sólo una vez). Además de los bloques lógicos, el *FPGA* posee interconexiones programables y bloques de entradas y salidas (*I/O*). En la actualidad, los *FPGA* también poseen procesadores, memoria interna, periféricos especializados para señales de reloj, entre otras cosas. Existen lenguajes de descripción de *hardware* (*HDL*) como *VHDL*, *Verilog* y *SystemVerilog* que se utilizan para alinear los bloques digitales, con las interconexiones programables y los bloques de entradas y salidas, que al ser descritos pueden formarse sistemas complejos. En cuanto a las aplicaciones, han ganado bastante importancia en sistemas de procesamiento digital, ya que según como se describa el *hardware*, son capaces de realizar funciones en paralelo y realizar operaciones complejas en pocos ciclos de reloj.

Microcontrolador *ARM* (*Advanced RISC Machine*)

La arquitectura *ARM* fue presentada en 1980 por la empresa *Acorn Atom*, desde sus inicios, esta arquitectura fue muy competitiva por tener un bajo consumo de energía. Dicha cualidad llamó la atención de varias de empresas como *Apple* que empezaron a diseñar productos basándose en esta arquitectura. Actualmente *ARM* es una empresa independiente, que posee más del 90% de las propiedades intelectuales de los dispositivos móviles del mundo. *ARM* tiene seis familias de arquitecturas de procesadores según las necesidades del sistema a diseñar, que se muestran a continuación:

- *CORTEX-A*, para el máximo procesamiento y consumo de potencia óptimo. Utilizada para celulares, capaz de correr sistemas operativos como *Linux* y *Android*.
- *CORTEX-R*, procesamiento confiable la ejecución para tareas críticas. Utilizada en aplicaciones médicas y automotrices, para cumplimiento de tareas relacionadas con *Functional Safety*.
- *CORTEX-M*, procesamiento para sistemas sencillos con el máximo ahorro de energía. Utilizado para aplicaciones con baterías para larga duración, como internet de las cosas, juguetes o aparatos de uso común.
- *Neoverse*, un sistema flexible para aplicaciones en la nube y para *edge computing*. Utilizado en servidores.
- *SecurCore*, diseñado para aplicaciones de seguridad en la capa física, lo que implica encriptación en protocolos de comunicación y chequeos para prevenir agujeros de seguridad.
- *Ethos*, diseñado para un alto desempeño en aplicaciones de *Machine Learning* e inteligencia artificial.

2.3 Máquinas Eléctricas

En la actualidad las máquinas eléctricas mueven al mundo. Sin embargo, hace aproximadamente siglo y medio, se buscaba un sistema de alta eficiencia capaz de transformar la energía eléctrica a energía mecánica que sustituyera las máquinas a vapor. Ese dispositivo se le conoce como motor eléctrico [11]. Existen varios tipos de motores eléctricos, donde su principal diferencia es su principio de funcionamiento. Una de las principales divisiones del motor, es si funcionaban con corriente directa o corriente altera [12]. Debido a su construcción, el principio de funcionamiento para moverlos es diferente. Es por eso, que para cada principio de funcionamiento se diseñan controladores con los cuales se puede tener un dominio de variables como velocidad, torque, aceleración, entre otros. Para los alcances de esta tesis, el enfoque será orientado exclusivamente a los motores de inducción trifásicos [13].

2.3.1 Motor Inducción Trifásico

El primer motor de inducción fue inventado por Nikola Tesla en el año de 1887. Un invento bastante revolucionario para la época, y no fue hasta un poco menos de 100 años después que el uso del motor empezó a popularizarse por sus cualidades en aplicaciones industriales. Dicho motor de inducción cuenta con varias ventajas sobre el motor de corriente directa, entre ellas se destaca que no tiene conmutador, ni escobillas, por lo que puede ser utilizado en un ambiente inflamable sin

riesgo de crear una chispa. Es más robusto , cuenta con menor inercia en el rotor, es de bajo mantenimiento y posee un torque elevado al partir del reposo. El principio de funcionamiento del motor de inducción consiste en tres bobinas posicionadas a 120° entre ellas en el estator. En medio se coloca una "jaula de ardilla", que consiste en dos anillos paralelos conectados por varias barras ligeramente inclinadas. Al inducir corriente alterna trifásica en las bobinas del estator, se producirá un campo magnético rotacional [14]. La corriente en cada bobina está dada por las ecuaciones 2.16, 2.17 y 2.18.

$$i_{as} = I_{s_m} cos\left(\omega t\right) \tag{2.16}$$

$$i_{bs} = I_{s_m} cos\left(\omega t - \frac{2}{3}\pi\right) \tag{2.17}$$

$$i_{cs} = I_{s_m} cos\left(\omega t - \frac{4}{3}\pi\right) \tag{2.18}$$

Donde I_{s_m} es la amplitud pico de las corrientes de entrada. Mismas, que dan como resultado del flujo de una corriente en el estator, donde se generaran vectores de fuerza magnetomotriz $\mathcal{F}_{as}$, $\mathcal{F}_{bs}$ y $\mathcal{F}_{cs}$. Que sumados dará el vector $\mathcal{F}_s$. Al mismo tiempo, la corriente generará líneas de flujo magnético en el estator, (Φ_s) y se generará una velocidad síncrona de los campos del estator esta dada por la ecuación 2.19. Donde p_p es el número de pares de polos. En la figura 2.6 se pueden observar las variables mencionadas en un modelo simplificado de de un motor de dos polos. La velocidad síncrona del motor también va a depender del número de polos [15].

$$\omega_{syn} = \frac{\omega}{p_p} \tag{2.19}$$

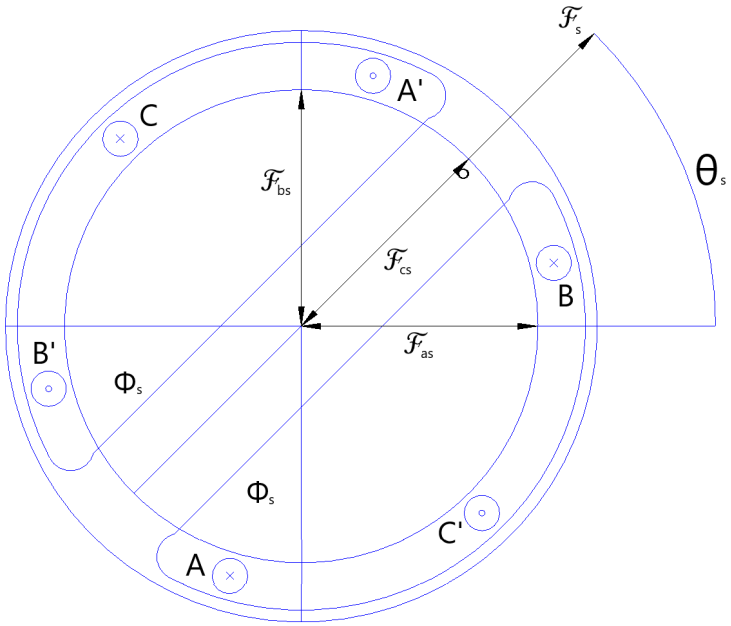


Figura 2.6: Diagrama fasores de las fuerzas magnetomotrices resultante en un motor de dos polos a 45°

Durante el estado estable, el rotor del motor trifásico lleva una velocidad inicial. Por otro lado, los conductores del rotor se ven impulsados por una campo magnético que es producido por las bobinas del estator, mismo que añade una velocidad que es mayor a la velocidad del inicial del rotor. La diferencia de estas dos velocidades induce una fuerza electromotriz, ya que si la velocidad del rotor y la velocidad del campo fueran iguales no existiría una fuerza electromotriz. Dicha fuerza electromotriz, va a producir una corriente en el conductor del rotor, que al interactuar con el campo magnético resulta en una fuerza electrodinámica, cuya dirección puede ser identificada fácilmente por la ley de la mano izquierda. Dentro de este mismo fenómeno se genera un torque, que esta definido por el radio del rotor y la fuerza electrodinámica [16].

Existe un fenómeno llamado deslizamiento, el cual esta presente cuando la velocidad del rotor es menor a la velocidad síncrona del motor. La diferencia de esas velocidades se le conoce como la velocidad de deslizamiento. El deslizamiento esta definido por la división de la velocidad de deslizamiento entre la velocidad síncrona del motor [17].

Circuito Equivalente

Para poder controlar un sistema es necesario modelarlo matemáticamente. Dado que un motor de inducción trifásico no es un sistema sencillo, han existido a lo largo de la historia varios modelos que asemejan su comportamiento.

Cuando se traba un rotor y se administra corriente a las bobinas del estator, estas tienen el mismo comportamiento de un transformador trifásico. Donde las bobinas las bobinas del estator

actúan como el embobinado primario y las del rotor como el embobinado secundario. A partir de esta analogía surge un modelo equivalente, que en la literatura se simplifica a través de análisis fasoriales y al mismo tiempo despreciando las perdidas mecánicas. Aunado a esto, la potencia de salida puede ser representada a través de una resistencia de carga. El circuito equivalente de cada fase del motor de inducción trifásico junto con la resistencia de carga se muestra en la figura 2.7. Donde R_s es la resistencia del estator, R_r es la resistencia del rotor, X_{fs} es la reactancia de fuga del estator, X_{fr} es la reactancia de fuga del rotor, X_m es la reactancia de magnetización [18].

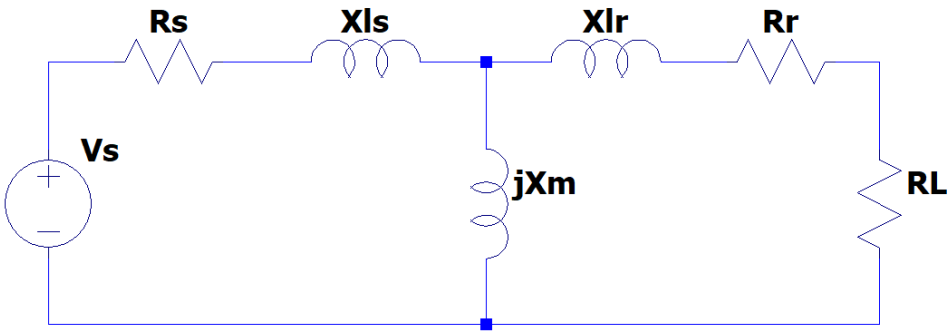


Figura 2.7: Circuito equivalente de una fase de un motor de inducci´on trif´asico, con una resistencia de carga.

2.4 *FPGA System on Chip (SoC)*

Actualmente son pocas las empresas que se dedican al diseño y fabricación de circuitos integrados *FPGA SoC (System on Chip)*, para el diseño de *software* y *hardware* en conjunto. Esto permite que la circuitería interna del circuito pueda ajustarse, al igual que el código. Esto facilita y mejora el proceso de diseño, además que minimiza las limitaciones que se tienen cuando se diseña sobre un circuito de alguna marca. Algunas compañías con este problema, dados los requerimientos tan específicos que tienen deciden diseñar un *ASIC* (*Application Specific Integrated Circuits*), a pesar del largo proceso de diseño y manufactura que implica, para ser competitivos en cuanto a costo. Sin embargo, con un *FPGA SoC* si existe limitante a nivel arquitectura interna, esta se puede solucionar y volver a programar las veces que sean necesarias.

Las dos empresas que más destacan son Intel y Xilinx, y son quienes se encargan de marcar la tendencia en esta tecnología. Dados los alcances de la tesis, el enfoque se hará en las soluciones de la marca Intel, antes Altera, específicamente en el *FPGA SoC* de la familia *Cyclone V*.

2.4.1 Arquitectura

El *FPGA SoC Cyclone V* esta constituido por dos etapas principales, el *FPGA* con 25,000 elementos lógicos y el *Hard Processor System (HPS)*, que contiene un procesador *ARM CORTEX-A9* de dos núcleos a 32 bits, con una frecuencia máxima de 925 MHz y un controlador para memorias *DDR3* a una frecuencia de 400 MHz. Ambas etapas están interconectadas por líneas de transmisión programables para poder compartir datos entre ellos.

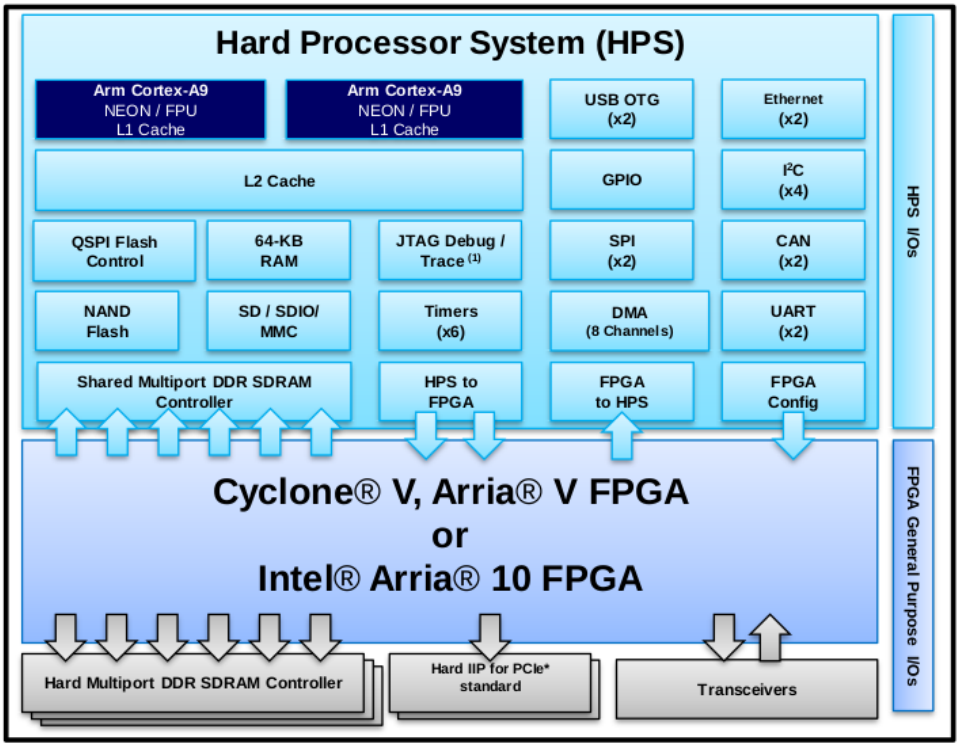


Figura 2.8: Arquitectura del *FPGA SoC Cyclone V*

Hard Processor System

Además de contener una unidad de procesamiento central (*MPU*) basada en la arquitectura ARM *CORTEX-A9*, que posee dos núcleos y un caché nivel dos. Las arquitecturas de ARM se caracterizan por tener líneas de transmisión que comunican a todos los subsistemas y periféricos que posee el circuito integrado, el protocolo de comunicación que siguen esos subsistemas se le conoce como *Advanced eXtensible Interface (AXI)*, mismo que se describirá más adelante. Cada núcleo cuenta con la tecnología *NEON*, que es una extensión de la arquitectura que permite manipular mútiples datos con una singular instrucción (*SIMD*), que facilita el procesamiento de señales, algoritmos de inteligencia artificial, o procesamiento de imágenes. Los núcleos también cuentan con un caché nivel 1 para instrucciones y para información, y una unidad escalar de punto flotante. Además los dos núcleos van acompañados de los siguientes subsistemas:

- Un *Snoop Control Unit (SCU)*, se encarga de mantener coherencia entre las memorias caché de ambos núcleos del procesador, además arbitra las peticiones de lectura al caché entre los procesadores e inicializa dichas peticiones provenientes de la línea *AXI* hacia el caché nivel 2.
- Un controlador de interrupciones genérico (*GIC*), para manejar las interrupciones tanto internas, como externas, inicializadas en el procesador.
- Temporizadores globales, para contabilizar los ciclos de ejecución del proceso, además de las tareas programadas.
- Temprorizador *Watchdog (WDT)*, se utiliza para garantizar los ciclos de ejecución y genera

un reinicio del sistema, en caso de no cumplir con ellos.

Finalmente, el *HPS* cuenta con múltiples periféricos que ayudan a interactuar con las interfaces externas que se utilizan para transmitir y recibir datos. El *Cyclone V* cuenta con los siguientes periféricos:

- 1 x Controlador de acceso directo a la memoria (*DMA*).
- 2 x Control de acceso a medios para *Ethernet* 10/100/1000.
- 1 x Controlador para memorias NAND (8 bits).
- 1 x Controlador para memorias *SD/MMC*.
- 1 x Controlador para comunicación *QSPI* (*Quad Serial Peripheral Interface*).
- 2 x Controlador para *USB* 2.0.
- 2 x Controlador para comunicación SPI.
- 1 x Controlador para comunicación *I2C* (*Inter-Integrated Circuit*).
- 2 x Controlador para comunicación *UART* (*universal asynchronous receiver-transmitter*).
- 3 x Controlador para entradas y salidas.
- 2 x Controlador para temporizadores y *WDT* (*Watchdog Timer*) de 32 bits.
- 2 x Controlador para comunicación *CAN* (*Controller Area Network*).

Además de los periféricos anteriores, un periférico muy importante que permite configurar el *FPGA*, y se cuenta con una interfaz para *debug* con el objetivo de detección de fallas. Este último va de la mano con los mecanismos de reinicio que posee el dispositivo, no sólo por *hardware* (pines específicos, *JTAG* (*Joint Test Action Group*), *WDT*), si no también por *software* (Linux).

FPGA

Dentro del *Cyclone V* el *FPGA* se utiliza como puente para administrar las interacciones de alta velocidad, entre los pines y los controladores internos. Esto significa que el primer paso que deberá realizar el procesador es configurar el *FPGA*. El controlador para memorias *SDRAM* es un buen ejemplo, ya que ocupa ocupa una secuencia de inicio, por lo que en caso de haber un sistema operativo, una calibración inicial deberá realizarse antes de usarse, y esta secuencia de inicio de describirá en la sección siguiente.

La conexión entre el *FPGA* y el *HPS* está definida por líneas de transmisión especializadas. El *Cyclone V* posee 4 líneas de transmisión entre los dos módulos principales, y son las siguientes:

- *Lightweight HPS to FPGA Bridge*, es una línea de transmisión de 32 bits para transmisiones sencillas y no soporta un tráfico constante de transacciones en la línea.
- *HPS to FPGA Bridge*, es una línea de transmisión de 64 bits específico para transmisiones y tráfico constante, del procesador ARM hacia el *FPGA*.
- *FPGA to HPS Bridge*, es una línea de transmisión de 64 bits específico para transmisiones y tráfico constante, del *FPGA* hacia el procesador *ARM*. Es la única línea que posee coherencia con el caché.
- Interconexión con *SDRAM*, es la línea de transmisión más rápida llevar información del *FPGA* a la memoria *RAM*, de ahí el procesador puede acceder a ella.

2.4.2 Comunicación entre Procesadores (*IPC*)

Como se explicó en la sección anterior, existen varias líneas de transmisión predeterminadas entre el *FPGA* y el *HPS*, estas líneas pueden formar un árbol de jerarquías de módulos maestros y esclavos que forman una red dentro del chip (*NoC*). Para poder mandar los datos y establecer las jerarquías, es necesario definir protocolos de comunicación, que ayuden a hacer nulas las pérdidas en la transmisión de datos y que nos den la posibilidad de tener un control en el flujo de información. Intel recomienda la implementación de un protocolo diseñado por ellos conocido como *Avalon* para todas las aplicaciones dentro del *FPGA*, de esta forma el diseño del sistema en el *FPGA* se debe diseñar por bloques que tengan jerarquías para la transmisión de datos. Por otro lado, las arquitecturas *ARM* sólo son compatibles con los protocolos de comunicación *AXI*, entonces se debe utilizar un método para intercomunicar ambos protocolos. Ambos protocolos se explican a continuación.

Avalon

Este protocolo de comunicación fue diseñado por Altera, ahora Intel, con el objetivo de hacer comunicación entre procesadores. Existen tres tipos de comunicación dentro del protocolo que dependen según la aplicación de cada bloque que se busca interconectar.

Avalon Streaming Interface Es un protocolo unidireccional, de punto a punto. El protocolo tiene una señal de reloj que lo hace síncrono y es usado para componentes que ocupan un ancho de banda grande y baja latencia, como para mandar información de un procesador de señales digitales (*DSP*). Para este protocolo se le llama *sink* al receptor y *source* al transmisor. Las señales involucradas en la transmisión se muestran en la figura 2.9. Donde las señales *startofpacket*, *endofpacket*, *channel*, *empty*, y *error* son opcionales para poder implementar el protocolo.

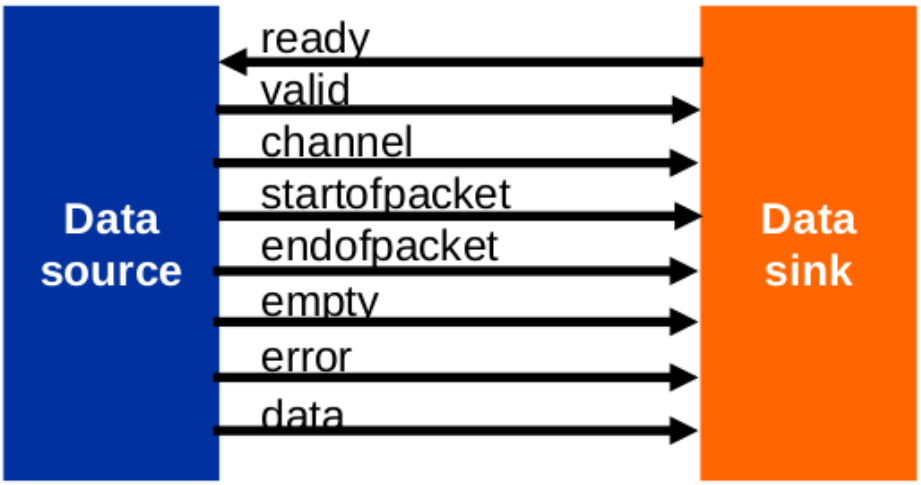


Figura 2.9: Bloque simplificado de comunicaci´on *Avalon Streaming Interface*

Avalon Memory-Mapped Interface Es un protocolo basado en direcciones, que permite a los dispositivos involucrados en la comunicación el leer y escribir usando peticiones para transferir datos. Las interfaces con el rol de maestro, tienen la capacidad de comunicarse con esclavos específicos iniciando peticiones a las direcciones de los esclavos. Los esclavos tienen la capacidad de recibir y escribir información de las peticiones apuntadas a su dirección, además de controlar el flujo de información con las señales de control. Dentro del proceso de transmisión, el esclavo puede controlar la latencia de forma fija o variable usando las señales de control. En cuanto al direccionamiento este suele ser de 32 bits y apunta a la dirección de memoria del esclavo, de modo que para pasar al siguiente registro se deberán sumar 4 bits a su dirección. Existen dos métodos avanzados para la transmisión de alta velocidad en este protocolo, el primero se le conoce como *pipelined*, y se utiliza cuando tanto esclavos como maestros deben realizar lecturas simultaneas. El modo *pipelined* permite iniciar nuevas lecturas antes que la lectura anterior haya terminado, sin embargo, este modo no puede ser utilizado para realizar escrituras a la línea de transmisión. El otro modo de operación avanzado se le conoce como *bursting*, este método permite al maestro tener una conexión con un esclavo de forma ininterrumpida por un número de transferencias consecutivas. Para ello, es necesario que el maestro indique al esclavo el número consecutivo de transacciones ininterrumpidas que se realizarán. Entre esclavos y maestros existe una interconexión que se encarga de hacer una distribución de señales entre los nodos involucrados en el sistema, por lo que las señales del esclavo y las del maestro no son las mismas, como se muestra en la figura 2.10.

Es importante recalcar que no todas las señales mostradas en la tabla 2.6 son utilizadas para realizar la comunicación mínima de este protocolo. Las señales mínimas son *address* y *waitrequest*, si se quiere escribir se deben añadir *write* y *writedata*, y para leer se deben añadir *read* y *readdata*. Además para el modo es necesario anexar la señal *waitrequest* y *readdatavalid*, mismas que controlan la latencia en estos modos. Por último, el modo *bursting* al ser de alta velocidad, ocupa las dos señales anteriores, y una señal *burstcount* para indicar el número de transacciones consecutivas.

Señales	**Tamaño**	**Dirección**	**Descripción**
address	1-64	Salida / Entrada	Señal usada para apuntar la dirección del esclavo deseado.
waitrequest	1	Entrada / Salida	Detiene al maestro de realizar envíos, sirve como control de latencia.
read	1	Salida / Entrada	Indica que el maestro esta realizando una lectura.
readdata	$2^3 - 2^{10}$	Entrada / Salida	Información de la lectura.
write	1	Salida / Entrada	Indica que el maestro esta realizando una escritura.
writedata	$2^3 - 2^{10}$	Salida / Entrada	Información de la escritura.
byteenable	$2^3 - 2^7$	Salida / Entrada	Especifica el número de bytes para leer o escribir.
lock / begintransfer	1	Salida / Entrada	Cuando el maestro inicia una transmisión con un esclavo, la señal se activa. / Indica el inicio de una transmisión nueva.
response	2	Entrada / Salida	Indica si la transmisión es exitosa, o si la dirección es válida.

Table 2.6: Señales involucradas en la interfaz de maestro / esclavo en *Avalon Memory-Mapped*.

Otros modos de *Avalon* Existen otros modos que también cubre el protocolo *Avalon* que son importantes, sin embargo no son tan complicados como los dos anteriores. *Avalon conduit* se utiliza para realizar de forma arbitraria una agrupación de puertos que van a ser exportados del bloque principal del sistema, este protocolo se utiliza cuando se direccionan las señales a pines de salida en el *FPGA SoC*. *Avalon interrupt* se ocupa para hacer un mapeo de las interrupciones en los bloques, para ello existe un remitente que genera una interrupción de un bit a un receptor. El receptor tiene la capacidad de recibir hasta 32 bits de interrupciones, y es capaz de identificar la fuente de la interrupción.

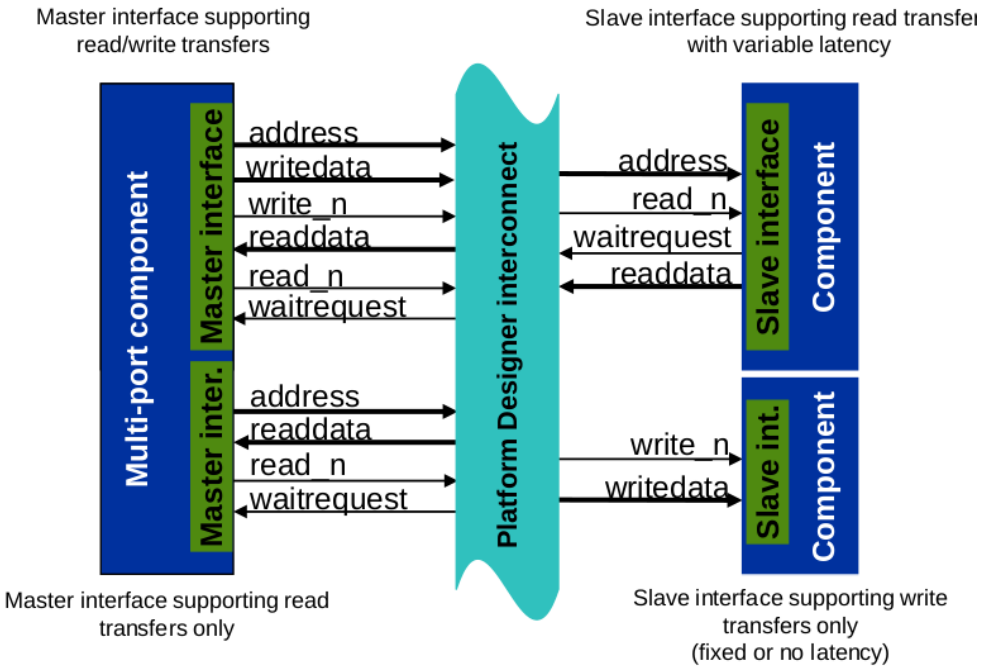


Figura 2.10: Bloque simplificado de comunicaci´on *Avalon Memory-Mapped Interface*

Arm AMBA AXI

Este protocolo de comunicación fue diseñado por *ARM* para comunicar los diversos periféricos dentro del procesador, de modo que fue diseñado pensando en anchos de banda elevados y baja latencia, operando a altas frecuencias. Provee diferentes parámetros para dirección, control y datos, que le permiten hacer transacciones consecutivas a alta velocidad. Además, posee métodos de transacción encriptados.

El protocolo de comunicación *AXI* posee 2 líneas de dirección, 3 líneas de datos, una línea de reloj y 2 líneas de control.

- Dirección: *read address*, y *write address*.
- Datos: *read data*, *write data*, y *write response*.
- Reloj: *CLK*.
- Control: *valid*, y *ready*.

2.4.3 Lenguaje de Descripción de *Hardware* (*HDL*)

VHDL contra Verilog

Los lenguajes de descripción de *hardware* surgieron en los años 80s. Primero surgió el *VHDL*(*VHSIC Hardware Description Language*) donde *VHSIC* significa *Very High Speed Integrated Circuit*, desarrollado por el departamento de defensa de Estados Unidos. Mientras que Verilog surgió como un desarrollo de *Gateway Design Automation Inc.* Estos dos lenguajes fueron los únicos que pudieron

adaptarse y tener el soporte para sobrevivir a la revolución digital que se ha tenido desde entonces. Es importante destacar que ambos lenguajes de descripción tienen la capacidad de realizar el mismo trabajo, y es muy extraño que alguien sostenga un argumento resaltando alguno de estos dos idiomas en cuestión de desempeño o eficiencia, puesto que eso depende del sintetizador y no del lenguaje. Sin embargo, la diferencia en cuanto a sintaxis es clara, y normalmente es el único punto de comparación que se utiliza para compararlos. Dichas diferencias provienen del origen de cada uno de ellos. El *VHDL* fue basado en el lenguaje ADA, que es proveniente del PASCAL, y Verilog fue basado en un *HDL* llamado HiLo, y complementado con el auge de C en esos años. Dada la popularidad que tuvo y aun tiene C, Verilog es más utilizado en la actualidad. Empresas como Intel, promueven el uso de Verilog para los desarrollos en su plataforma, aunque también brindan soporte para *VHDL*. Ciertamente, *VHDL* es más descriptivo y detallado al Verilog, lo que no parece ser de agrado para varios desarrolladores, puesto que Verilog es mucho más rápido de implementar.

Como la revolución digital esta en proceso y la ley de Moore sigue siendo vigente, a pesar de estar cerca de romperse. Las necesidades en cuanto a diseño digital han presentado una nueva opción llamada *SystemVerilog* que se ha adaptado a los requisitos del desarrollo actual y simplifica el diseño de descripción de *hardware*. Además que presenta un acercamiento similar a la programación orientada a objetos y posee herramientas para la validación de *hardware* para los modelos diseñados, algo un poco tedioso de hacer con *VHDL* y Verilog; se podría considerar un híbrido. Este nuevo lenguaje de descripción esta teniendo buena aceptación de las principales compañías de diseño digital, por lo que se muestra prometedor para un futuro cercano [19].

2.5 Linux

Linux es un sistema bastante extenso y complejo, que para la gente que no estudia los sistemas computacionales, puede parecer un poco intimidan te al inicio. Sin embargo, el aprendizaje de Linux debe ser abordado a partir de abstracciones con complejidad incremental. Esto significa que los niveles de abstracción, donde el nivel más bajo es la capa física que incluye el procesador, las memorias, y periféricos; seguido de la segunda capa que sería el *kernel*, que integra todo el *hardware* en un sistema y administra los recursos para poder explotar el potencial de los recursos de la capa anterior. Posterior a el *kernel*, esta el espacio de usuario que incluye la terminal donde el usuario puede interactuar con el sistema, y va a escalando hasta el nivel de aplicación como lo son los navegadores o juegos de vídeo. Para los alcances de esta tesis, se analizaran a detalle las capas de más bajo nivel [20].

Dadas las tendencias de dispositivos móviles e internet de las cosas generaron las necesidades de adoptar un sistema que fuera ubicuo, de bajo costo, robusto y durable. Poco a poco, los diseños de *SoC* han empezado a desplazar los microcontroladores para aplicaciones de sistemas embebidos, dada su capacidad de poder adoptar Linux embebido en un solo circuito integrado o con muy pocos componentes externos. Así como Linux en su versión para computadoras o servidores existen varias distribuciones de Linux embebido. Para los alcances de esta tesis, el enfoque sera sobre la distribución del proyecto Yocto, que es un conjunto de proyectos dedicados a integrar un *build system* como *OpenEmbedded* para poder hacer una compilación cruzada con la *metadata* del dispositivo seleccionado, y así poder correr Linux en él. El proyecto Yocto usa capas de integración para poder definir las jerarquías de ejecución que soportan Linux. La capa *BSP (Board Support Package)*, contiene los parámetros para personalizar los cambios que se hagan al proyecto Yocto,

para adaptarlo a nuestro *hardware*. Esas capas que se integran a Yocto comienzan con el prefijo `meta`, que define la *metadata* con nuestros cambios, como lo es `meta-bsp-custom`. Los elementos para poder integrar Linux se muestran a continuación [21].

2.5.1 *Toolchain*

El *toolchain*, es la base para compilar todo el código que será implementado en el dispositivo, y deberá ser capaz de optimizar las instrucciones para la arquitectura del procesador. Inicialmente, se utiliza para construir el *bootloader*, el *kernel* y los archivos del sistema base. El *toolchain* esta formado por componentes del proyecto GNU, entre los cuales destacan: *Binutils*, un conjunto de utilidades binarias incluyendo el ensamblador y el *linker*; *GNU Compiler Collection (GCC)*, el compilador de C y otros lenguajes de programación; la librería de C, una *API (Application Program Interface)* que es la principal interfaz entre el sistema operativo y las aplicaciones.

Existen dos tipos de *toolchains*, la nativa, donde el sistema es del mismo tipo que los programas que genera, como suele ser en computadoras de escritorio y servidores. Y la tipo *cross*, donde el sistema donde se genera es diferente al sistema objetivo de los programas que genera, usualmente utilizado en computadoras para compilar programas para sistemas embebidos. Además para la construcción de un *toolchain* se deben considerar factores como la arquitectura del sistema, si las operaciones son *big-endian* o *little-endian*, si posee unidad de punto flotante, la convención usada para pasar parámetros entre funciones, entre otros. Para esta tesis, el *toolchain* propuesto es *Linaro*.

2.5.2 *Bootloader*

El *bootloader* es el encargado de inicializar el sistema y cargar el *kernel* del sistema operativo usando la estructura descrita en el árbol de dispositivos *(device tree)*, el cual se explicará más adelante. El *bootloader* es ejecutado acto seguido de alimentar el circuito, o tras un reinicio. En ese momento, ningún periférico o memoria ha sido inicializado, por lo que debe ser ejecutado desde una memoria interna. Su propósito para el final de la ejecución es iniciar los periféricos y cargar el *kernel* de Linux en la memoria RAM para alistar el entorno de ejecución. Una vez que el *kernel* inicia su ejecución, puede tomar toda la memoria que utilizó el *bootloader*.

Actualmente, la herramienta *opensource U-Boot* es la más popular de *bootloader* en sistemas embebidos. Esta herramienta comenzó siendo utilizada para arquitecturas *PowerPC*, y posteriormente, adoptó arquitecturas basadas en *ARM* y *MIPS* (*Microprocessor without Interlocked Pipelined Stages)*). A raíz de su popularidad cuenta con más de mil archivos de configuración para diversas tarjetas con sistemas embebidos. Al momento de construir los archivos del *bootloader* con esta herramienta es fundamental establecer el *toolchain* correcto, de esa forma se puede hacer el *cross-compile* de forma correcta.

2.5.3 Kernel

El propósito principal del *kernel* de Linux es interactuar con el *hardware* de la computadora, implementar una interfaz de operación y control programable con los componentes, programar el acceso a los recursos a nivel *hardware*, y proveer un entorno de ejecución para programas del usuario, que sea fácil de usar. El *kernel* de Linux es monolítico, lo que significa que se corre sobre un solo hilo de ejecución. Esto es una ventaja, puesto que el código del kernel es compacto y rápido, sin embargo, carece de jerarquías. Este tipo de *kernel* posee tres niveles de jerarquías que definen su ejecución como se ve en la figura 2.11. Donde el nivel superior esta definido por la capa del

programa principal, que llama a un servicio; el servicio hace una llamada al sistema; y las funciones brindan soporte a esa llamada al sistema [22].

El *kernel* posee 5 módulos principales:

- Módulo de organización de procesos, es el encargado de controlar el uso de los recursos de procesamiento para cada proceso, asegurando el cumplimiento periódico de cada tarea.
- Módulo de administración de memoria, es el encargado de administrar el uso de memoria *RAM*, de forma segura, entre los procesos. Además se encarga de soportar la virtualización en la memoria para que el sistema operativo no utilice más memoria de la que se tiene.
- Módulo de almacenamiento del sistema, es el encargado de controlar las transacciones con las memorias externas utilizadas en el sistema. Como pueden ser memorias *flash*.
- Módulo de comunicación entre los procesos, como su nombre lo indica, soporta el intercambio de información entre dos procesos en ejecución.
- Módulo de interfaces de red, es el encargado de implementar los protocolos de comunicación que pueden soportar los periféricos del sistema.

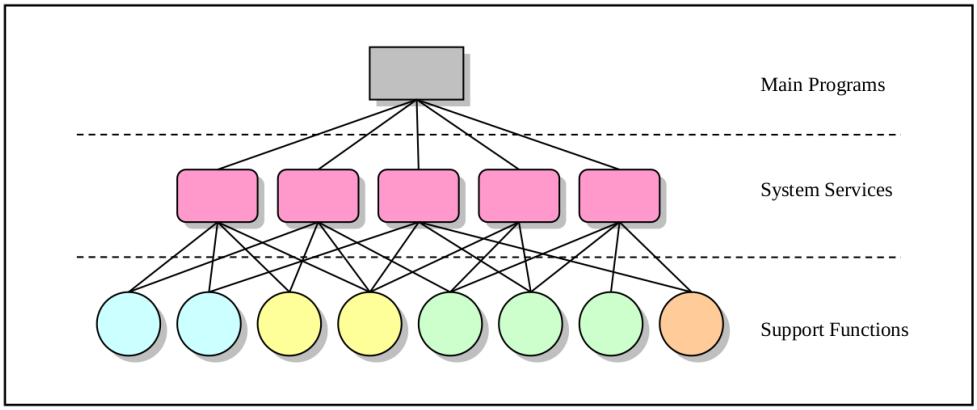


Figura 2.11: Estructura simplificada del *kernel* de Linux.

Espacio de Kernel y Espacio de Usuario

Se le conoce espacio de usuario a todos los procesos y aplicaciones en el sistema operativo que están fuera del *kernel*. Toda la información generada y utilizada por esos programas, es procesada y acumulada en memorias externas. La manera en el que el espacio de usuario tiene acceso a dicho espacio de memoria, es a través de peticiones al kernel, llamadas *system calls*. Por otro lado, el espacio de *kernel* cuenta con acceso total a cualquier espacio de memoria y cualquier periférico, por lo que sólo se utilizan las funciones categorizadas como confiables para su ejecución, y garantizar la integridad del sistema. De modo que si una función del *kernel* falla, el sistema puede tener un error fatal.

La importancia de este tema radica en que en un sistema embebido de tipo *FPGA SoC*, dada la naturaleza del sistema para reconfigurar el hardware interno del circuito, complica la implementación de aplicaciones a nivel usuario por las limitantes de acceso que tiene a nivel *kernel*. Al

momento de modificar el hardware a través de un cambio de configuración en el *FPGA*, se suele definir una nueva dirección de acceso a través del puerto *Avalon* para cada bloque. Además que el acceso del espacio de usuario a las líneas de comunicación está restringido. Se debe buscar la manera de poder interactuar entre ambos, sin afectar el funcionamiento del sistema operativo. La solución a este problema, es por medio de controladores.

Modelo de Controladores de Dispositivos

El modelo de controladores de dispositivos fue una implementación en el *kernel* de Linux a partir de la versión 2.6. A partir de ese momento, se introdujo un mecanismo universal para organizar los dispositivos en las líneas de transmisión. El mecanismo es capaz de determinar todos los dispositivos conectados, y sus cambios de estado, así como la capacidad de asignar un controlador válido para los dispositivos que se conecten. Un controlador es un *software* que tiene funciones para manipular un dispositivo que se conecte a una línea de transmisión. El modelo propone tres estructuras de datos principales. La estructura para representar un protocolo de comunicación, `struct bus_type`. La estructura para representar un controlador de un dispositivo, `struct device_driver`. La estructura para representar un dispositivo conectado a un protocolo de comunicación, `struct device`.

Árboles de dispositivos Son una estructura de datos para describir el *hardware*, y que el sistema operativo pueda interpretar el uso del dispositivo de forma sencilla. Cada elemento del árbol de dispositivos se le conoce como nodo, y cada nodo tiene propiedades para encapsular la información. Existen algunas categorías predefinidas para elementos comunes como lo son líneas de datos, líneas de interrupciones, resignación de puertos de entradas y salidas, dispositivos en los periféricos, entre otros. De esa forma se puede reutilizar el código para darle soporte al nodo, y en caso de necesitar alguna extensión se puede implementar, o incluso crear un tipo de categoría desde cero. La información de los árboles de dispositivos esta definida por un documento de texto, con terminaciones `.dtsi`(compatible entre plataformas) o `.dts`(específico de la plataforma). Y tienen tres propósitos principales que son:

- Identificación de la plataforma. Para que el *kernel* tenga la capacidad de reconocer la plataforma sobre la que se va a correr, todo esto durante la inicialización. Esto con el objetivo que el *kernel* haga los ajustes necesarios para poder correr, ya que hay variaciones de arquitecturas en el *CPU* o el *SoC*. Por ejemplo, en el caso de la arquitectura *ARM*, la función de inicio `setup arch()`, mandará a llamar una librería ubicada en `arch/arm/kernel/devtree.c`, donde se encontrará el parámetro `machine_desc` que determina la compatibilidad de la arquitectura y redirigirá al sistema para encontrar el árbol de dispositivos más compatible en `arch/arm/boot/dts`. En caso de no encontrarlo la función regresará un valor nulo y no inicializará los dispositivos de forma correcta, o no iniciará en absoluto.
- Configuración del tiempo de ejecución. Normalmente, el árbol de dispositivos es el único método para comunicar el *firmware* con el *kernel*. Al mismo tiempo, también es el encargado de proveer los parámetros de la configuración del tiempo de ejecución, como lo es la dirección de la memoria *RAM* donde inicia el sistema (`initrd`). Esos parámetros se inician en el nodo de nombre `chosen`.
- Población del dispositivo. Una vez que se identificaron los elementos definidos en el árbol de dispositivos, y que se definieron los parámetros de inicialización, el kernel procede a operar de

forma nominal para inicializar el sistema operativo. Pero antes, es necesario ver su compatibilidad con la plataforma donde se van a ejecutar. A esto se le conoce como popular, y lo que se hace es crear un dispositivo en la plataforma por cada nodo en el árbol de dispositivos, y guardarlo en la memoria para poder utilizar cada dispositivo de forma correcta durante toda la operación.

2.5.4 *Root filesystem*

Este es el último elemento necesario para correr Linux, y comienza a ejecutarse una vez que el *bootloader* cargó de forma correcta el kernel. El primer objetivo es crear un *root filesystem* mínimo, que de como resultado una terminal. Posteriormente, se pueden añadir *scripts* para inicializar otros programas, configurar la interfaz de red, configurar los permisos de los usuarios, entre otras cosas. Los componentes mínimos en un *root filesystem* son los siguientes:

- `init`: Es el programa que inicia todo, a través de una serie de *scripts*.
- *Shell*: Es el interpretador de comandos. Un programa que corre programas, necesario para la inicialización y uso de terminal.
- *Daemons*: Es un programa que corre en segundo plano, que provee servicios a otros programas. Los *daemons* iniciales son ejecutados por el comando `init`, que también se le podría considerar como uno.
- Librerías compartidas: Múltiples programas están ligados a través de librerías compartidas, por lo que deben estar presentes en el sistema.
- Archivos de configuración: La configuración del comando `init` y de los *daemons*, debe estar guardada en un archivo de texto en la dirección `/etc`.
- Nodos de dispositivos: Son archivos especiales que dan accesibilidad a varios controladores de dispositivos.
- `/proc` y `/sys`: Poseen estructuras del *kernel* en forma de carpetas y archivos, que son necesarios para el funcionamiento de varios programas.
- Módulos del *kernel*: En el caso de haber configurado algunas partes del *kernel* como módulos, estos deberán ser instalados en el *root filesystem* en la dirección `/lib/modules/[versión del kernel]`.

El sistema operativo no se ve afectado por la organización de los directorios del root filesystem, por lo que existen variaciones entre las distribuciones. Sin embargo, normalmente se el acomodo definido por la *Filesystem Hierarchy Standard (FHS)*. Para sistemas embebidos el esqueleto de directorios normalmente utilizado posee las siguientes carpetas:

- `/bin`: Para programas esenciales para todos los usuarios.
- `/dev`: Para todos los nodos de dispositivos y periféricos.
- `/etc`: Para archivos de configuración del sistema.
- `/lib`: Para librerías compartidas esenciales, como `stdio.h`.
- `/proc`: Un pseudo archivo que no existe en el disco, pero el kernel usa este espacio de memoria para proveer información sobre el *hardware* y los módulos de *kernel*.

- `/sbin`: Para programas esenciales para el administrador.
- `/sys`:
- `/tmp`: Para archivos temporales y volátiles.
- `/usr`: Para programas adicionales, librerías, y archivos del usuario.
- `/var`: Establece una jerarquía de archivos y directorios que pueden ser modificados durante la operación del sistema.

Capítulo 3

Metodología

3.1 Sistematización

La comprensión de un fenómeno se simplifica cuando se parte de una perspectiva general, a nivel sistema. Para posteriormente, analizar los componentes a un nivel más específico y comprender la interacción entre los mismos.

El sistema simplificado en la figura 3.1 comienza por el motor de inducción trifásico que se pretende controlar, mismo que estará instrumentado con sensores de tipo analógico y digital para poder monitorear las variables del mismo. Las señales de los sensores serán procesadas por el *FPGA SoC*, mismo que generará las señales de control que accionarán el inversor. Al mismo tiempo, el *FPGA SoC* podrá ser accionado y monitoreado a través de una red local, sobre la cual se generará un tráfico de información. Dicha información será procesada por una computadora, con la cual el usuario podrá observar el comportamiento de las variables del motor, así como del accionamiento de los comandos para el control de velocidad.

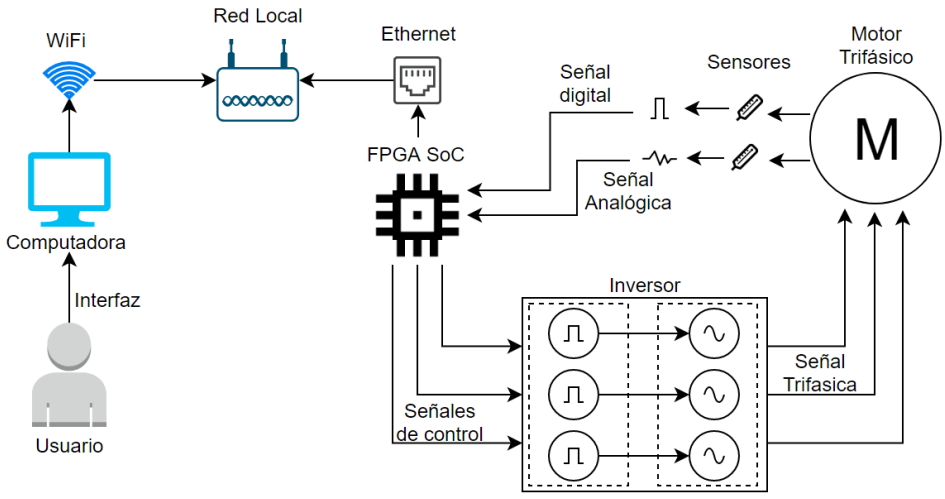


Figura 3.1: Abstracci´on a nivel sistema de los componentes principales y sus interacciones

Para segmentar dichos componentes, con el objetivo de realizar un análisis más exhaustivo sobre cada elemento, el sistema fue dividido en las siguientes etapas:

1. **Motor**: El motor definido para probar el inversor es un motor de inducción trifásico, TEFC, para trabajos pesados de la marca "Motors US", número de serie G04-AM55-M y modelo AM55. El motor tiene una potencia de $0.5hp$ y una velocidad nominal de de $1800rpm$. Su voltaje de entrada puede variar de $208V$ a $460V$. Dichos motores se caracterizan por por su buen desempeño en ambientes hostiles, en aplicaciones como bombas, compresores o ventiladores. Son totalmente cerrados con ventilación, para resistir la humedad o el polvo.
2. **Etapa de monitoreo y potencia**: Esta etapa consta de la circuitería necesaria para aislar las señales de control y convertirlas en señales de alta tensión que serán suministradas al motor; medir las señales de temperatura, corriente y voltaje suministradas al motor, junto con toda la circuitería para acoplar dichas señales que entrarán a un convertidor analógico digital paralelo, de alta velocidad.
3. **Etapa de *VLSI*/Linux**: Esta etapa cuenta con un procesador de dos núcleos y un *FPGA*. El primero se encargará de guardar y transmitir los datos a una red local, y de recibir los parámetros e instrucciones de control. El segundo se encargará de traducir las instrucciones de control a señales eléctricas y de procesar los datos de entrada provenientes de los *ADCs* y del *GPIO* (*General Purpose Input/Output*).
4. **Etapa de Software**: Esta etapa cuenta con una interfaz amigable con el usuario para poder observar los parámetros censados por la etapa de monitoreo, a través de gráficas. Igualmente, será capaz de transmitir los parámetros de control deseados al motor.

Las primeras cuatro etapas conforman el sistema de control de velocidad del motor con la estructura que se muestra en la figura 3.2.

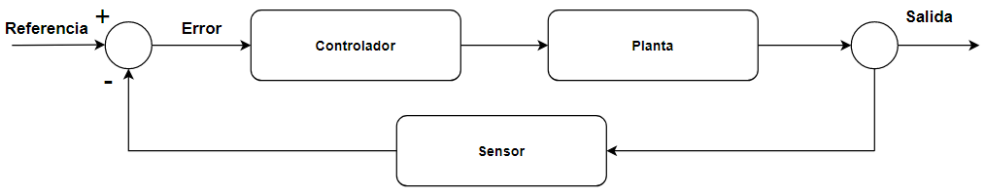


Figura 3.2: Estructura b´asica de un sistema de control en lazo cerrado.

En términos de este objeto de estudio, las partes del sistema de control en lazo cerrado quedan definidas de la siguiente forma.

- **Planta**: La planta a controlar será un motor de inducción trifásico.
- **Sensor**: Los sensores para el control del motor, serán sensores de corriente de efecto Hall, y un sensor de velocidad de tipo *encoder*, a través de un emisor y un receptor infrarrojos.
- **Controlador**: El controlador será un *FPGA SoC*, mismo que estará conectado a un inversor y a una tarjeta de adquisición de datos para poder acoplar las señales a las demás etapas.

Finalmente, se procede a definir las capacidades(límites) del sistema, para poder diseñar las etapas con tecnologías y componentes que puedan cumplir con dichas especificaciones.

3.2 Diseño

3.2.1 Etapa de Monitoreo y Potencia

Capacidades

Las capacidades eléctricas que debe cumplir el sistema son las siguientes:

- Voltaje directo de entrada máximo - $300V$.
- Potencia máxima de salida - $500W$.
- Frecuencia de conmutación máxima - $10kHz$.
- Número máximo de fases - 3ϕ.

Diseño de Potencia

Inversor Dado que el motor de inducción trifásico ocupa una señal alterna y el voltaje de entrada es directo, el convertidor de energía adecuado será un inversor de tres fases. La arquitectura básica de un inversor de este tipo se muestra en la figura 3.3.

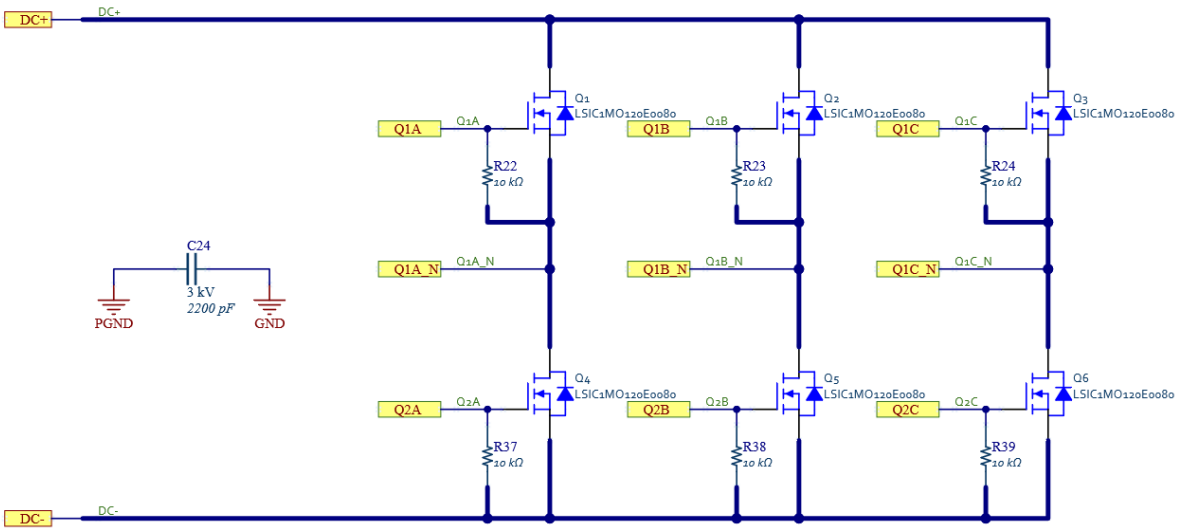


Figura 3.3: Puente H trif´asico, con transisores tipo FET con tecnolog´ıa de carburo de silicio. A la izquierda, un capacitor para desacoplar la tierra de potencia y la tierra digital.

Consta de un voltaje de entrada de corriente directa, que pasa por un capacitor para filtrar la entrada y para oponerse a las variación de tensión abruptos que pueda tener el sistema. Posteriormente, el flujo de corriente es controlado por 6 elementos conmutadores, para este inversor se utilizarán *MOSFETs* (*Metal-Oxide Semiconductor Field-Effect Transistor*). Para cumplir con los requerimientos de voltaje de entrada y de frecuencia de conmutación, se deben elegir partes que soporten dichas condiciones . Para ello, los *MOSFETs* serán de la tecnología SIC (Carburo de silicio), LSIC1MO120E0080,un material reciente utilizado por empresas automotrices en automóviles

eléctricos. En el caso de los capacitores, una regla de dedo es que el voltaje de tolerancia debe ser por lo menos 30% más del voltaje máximo calculado. Adicionalmente, se añaden unas resistencias de sangrado, para disipar la carga de los capacitores cuando el dispositivo sea desconectado. También, se coloca una resistencia de 10kΩ para garantizar que los dispositivos estén apagados.

Impulsores Para poder garantizar la activación de los *MOSFETs* en la configuración de la figura anterior, es necesario utilizar un método para flotar una tierra virtual para siempre garantizar el voltaje entre la fuente y la compuerta (VGS). El impulsor seleccionado para esta aplicación es el ISO5852SDWR de la marca Texas Instruments. Esta parte provee aislamiento de hasta $5.7kVrms$, y es capaz de controlar dos salidas complementarias con un tiempo de propagación de $76ns$. Posee detección en caso de sobre-corriente que activa una protección que deshabilita los pines de salida y reporta el error como señal de fallo. Esta detección de sobre-corriente posee un comparador interno que va a comparar el voltaje de des-saturación en la fase. El voltaje de referencia del comparador son 9V, pero se puede ver modificado al ponerle un diodo Zener en serie, para aumentar el rango. De igual forma, se coloca un diodo Zener en paralelo a un capacitor para proteger el circuito integrado en caso de un pico de voltaje mayor al aceptado. La implementación del impulsor ISO5852SDWR, se muestra en la figura 3.4, mismo que esta presente dos por fase, dando un total de seis en el circuito final.

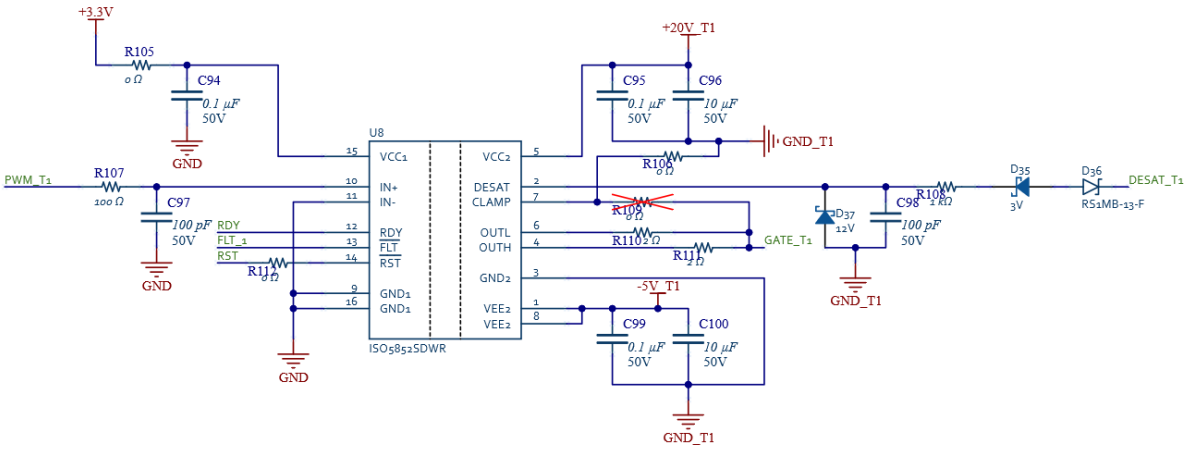


Figura 3.4: Configuraci´on para impulsor de media fase.

Al mismo tiempo, esta etapa es ideal para aislar las señales de control que irán al motor, el impulsor ya cuenta con esta característica de aislamiento interno en el circuito integrado. Lo único que se debe garantizar es el voltaje de entrada aislado en las fuentes de alimentación del impulsor, mismos voltajes serán definidos por los datos del *MOSFET* a activar.

Fuente de alimentación aislada Para dicha fuente de alimentación, se utiliza un transformador de pulsos operando a $500kHz$, que posee un rectificador de *tap* central. Posteriormente, la señal se filtra a través de múltiples capacitores cerámicos y entra a una configuración de un diodo Zener ajustable, para garantizar que el voltaje negativo (que será igual al voltaje recomendado para apagar el *MOSFET* de carburo de silicio) y la tierra mantengan el voltaje deseado en todo momento. Es

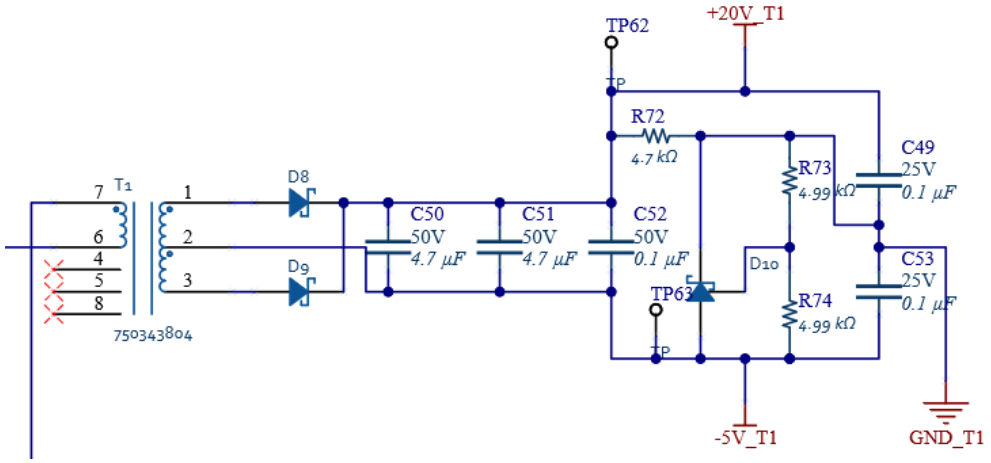


Figura 3.5: Configuraci´on para aislamiento de impulsores con transformadores de pulso.

importante destacar que dicha fuente de alimentación sólo alimentará el circuito del impulsor, por lo que su corriente de consumo es baja. La implementación de la fuente de voltaje aislada, se muestra en la figura 3.5.

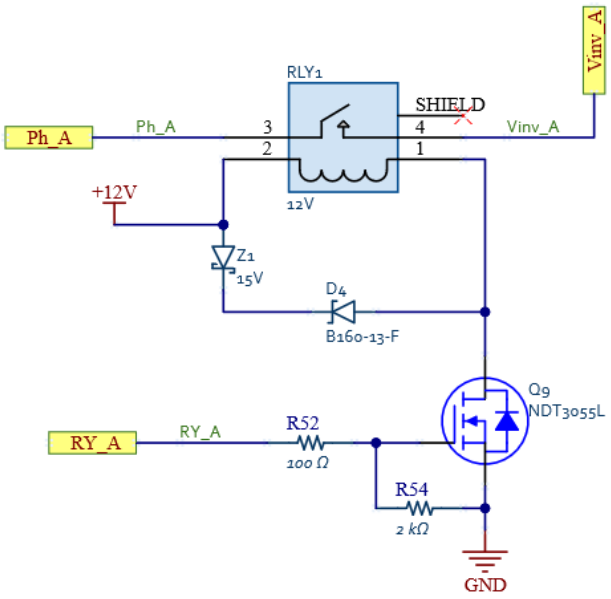


Figura 3.6: Configuraci´on para conmutaci´on r´apida de relevadores de protecci´on.

Relevadores Se decidió como medida de precaución poner un circuito de relevadores como una segunda protección a sobre corrientes. El circuito es activado a través de una señal digital que entra

a la compuerta del *MOSFET*. Cuenta con un arreglo de diodos en serie, cuya función es la rápida descarga de la carga acumulada en la bobina del relevador para una conmutación instantánea. De igual forma, se pone una resistencia de $2k\Omega$ para garantizar que el *MOSFET* se apague al poner un voltaje bajo en la compuerta. La implementación del relevador se muestra en la figura 3.6.

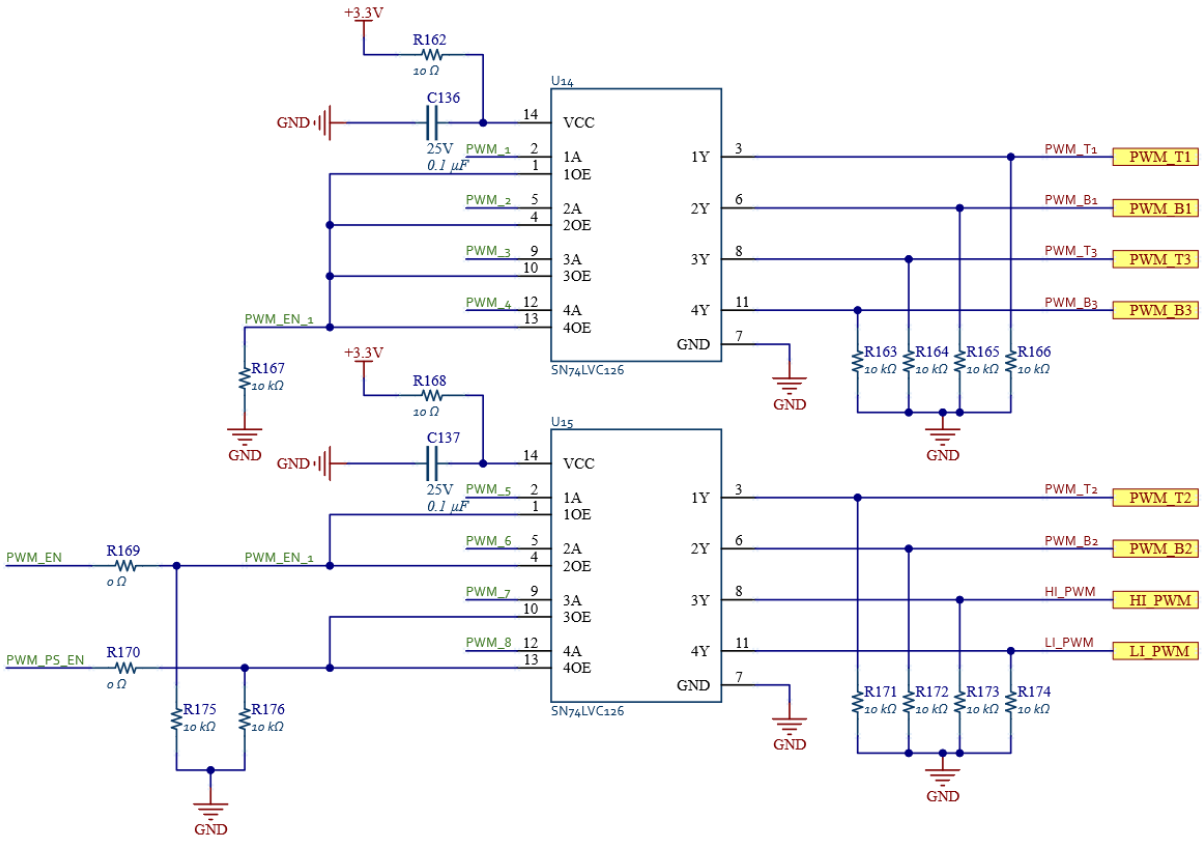


Figura 3.7: Circuito integrado con *buffers* para el aislamiento de las se˜nales digitales.

Buffers Los *buffers* fueron implementados con la función de proteger las salidas del procesador digital, haciendo que las salidas del *buffer* varíen entre un estado alto y alta impedancia. Cuando la salida del *buffer* esté en estado de alta impedancia, se tendrá un valor de salida equivalente a un cero lógico, por las resistencias de *pull-down*. Las entradas tendrán señales de pulsos con modulación que activarán un medio puente H (para los transformadores de pulsos) y además controlarán la conmutación de las fases del inversor. La implementación del *buffer*, se logra con el circuito integrado SN74LVC126 y se muestra en la figura 3.7.

Diseño de Monitoreo

Los circuitos que se presentaron en la sección anterior van a controlar la activación del inversor para poder controlarlo, en esta etapa vamos a introducir una serie de circuitos utilizados para leer las variables físicas del motor. Las variables de interés son el voltaje de la fuente de corriente directa,

el voltaje de cada fase, la corriente de cada fase, la velocidad del motor y el torque. El circuito está protegido para ser alimentado con un rango de voltaje de entrada para la parte lógica de baja potencia es entre 12V y 36V. Sin embargo, el circuito tiene la posibilidad de saltarse esta etapa si el voltaje de entrada está regulado a 12V. El regulador de voltaje seleccionado es de *Texas Instruments*, y su número de parte es LM61440. Un regulador de tipo *Buck* de alto desempeño, y bajo ruido. Diseñado para minimizar la interferencia electromagnética. Para el correcto funcionamiento de la parte, es necesario tomar en cuenta los siguientes parámetros:

- Definir la frecuencia de conmutación de la fuente, la parte es capaz de variar dicha frecuencia a través de una resistencia (RRT), en un rango de $200kHz$ y $2.2MHz$, justo en las frecuencias superiores e inferiores a la banda de AM (Amplitud Modulada). Una cualidad fundamental para el desempeño del circuito cuando es sometido a pruebas de *EMC* (*Electromagnetic Compatibility*).
- Se debe colocar un capacitor de *bootstrap* en el pin CBOOT, que se utilizará como un regulador *charge pump*, en caso que el voltaje VBOOT baje de 2.1V, para garantizar el funcionamiento de conmutación de la fuente.
- Definir el voltaje de salida y poner el divisor de voltaje correspondiente entre la salida, después del inductor y tierra. Dicha señal de retroalimentación se conectará al pin de FB. Dichas resistencias deben ser entre $10k\Omega$ y $1M\Omega$.
- Asegurar que la carga sea suficiente para que el dispositivo opere en *CCM* (*Continuous Current Mode*). Dado que en bajas corrientes funciona en *DCM* (*Discontinuous Current Mode*), con el objetivo de ahorrar energía.
- Para la selección del inductor, el valor de inductancia depende de los voltajes de entrada y salida, la frecuencia de conmutación y de la corriente máxima de salida.
- Finalmente, la selección del capacitor de salida depende de dos factores, la capacitancia y la resistencia equivalente en serie (ESR). Ambos factores determinarán el voltaje de rizo y la respuesta de la fuente a transientes. Si la capacitancia es muy grande, la fuente no será capaz de responder rápidamente a variaciones abruptas en la carga de salida; si la capacitancia es baja, el voltaje de rizo será muy elevado y generará mucho ruido a los componentes que vaya a alimentar [23].

Para la fuente de alimentación de $5V$, se decidió usar un módulo pre-ensamblado de *Texas Instruments*, con número de parte PTH08080WAD. Eso simplifica el diseño de dicha fuente conmutada, con muy pocos componentes. El voltaje de salida es definido con una sola resistencia. Se ponen capacitores de filtrado en la entrada y salida, y una ferrita en la entrada para filtrar elementos de alta frecuencia. De igual forma, un diodo emisor de luz para tener una indicación que existe voltaje de salida. La implementación del circuito se muestra en la figura 3.9.

El voltaje de $3.3V$ es regulado por una fuente de tipo lineal comercial, TLV1117LV33, con el objetivo de tener una salida de voltaje más limpia, sin componentes de alta frecuencia que podría tener una fuente conmutada. Además el número de componentes para la aplicación es bajo y el diseño de las mismas es muy sencillo, dado que la corriente que va a consumir esa línea de voltaje es muy baja. De igual forma, un diodo emisor de luz para tener una indicación que existe voltaje de salida. La implementación del circuito se muestra en la figura 3.10.

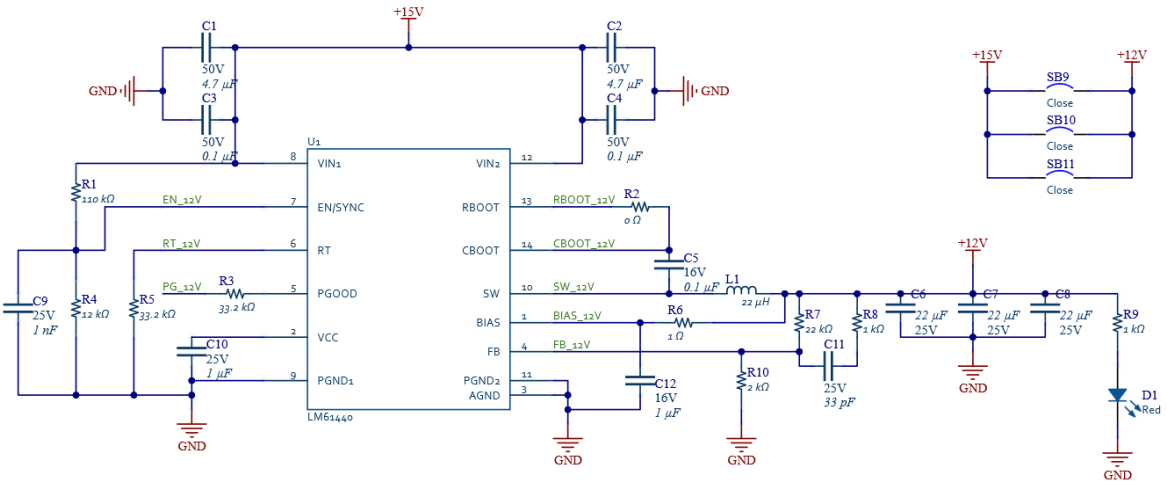


Figura 3.8: Fuente de alimentaci´on conmutada para regular el voltaje de entrada a la tarjeta a +12V.

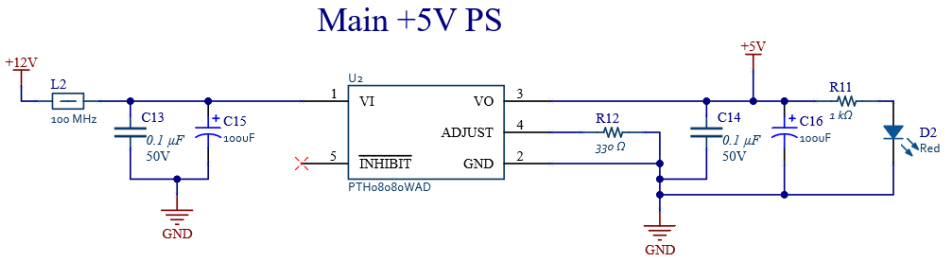


Figura 3.9: Fuente de alimentaci´on conmutada para regular el voltaje de +12V a +5V.

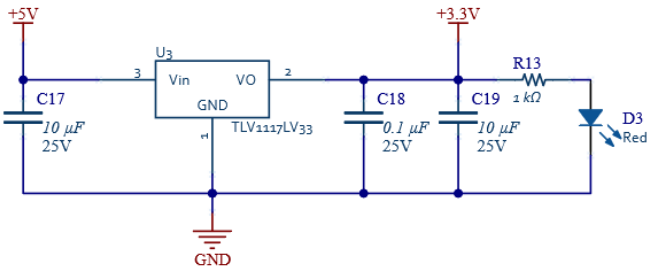


Figura 3.10: Fuente de alimentaci´on lineal para regular el voltaje de $5V$ a $3.3V$

Sensores de Corriente Para mantener el aislamiento, el método elegido para medir corriente es por medio de sensores de efecto Hall, donde se cuantifica el campo electromagnético producido por una corriente sobre un conductor. Para esta aplicación, se eligieron sensores de la marca LEM, con número de parte CASR15-N. Este tipo de sensores son ampliamente utilizados por los altos niveles

de corriente que se pueden alcanzar a medir, independientemente del voltaje que fluya a través de la línea. Con una precisión de $\pm 0.8\%$ y sensibilidad de $41.67mV/A$. La parte seleccionada alcanza a medir corrientes máximas de $15A$. La salida del sensor entra a un amplificador operacional en configuración diferencial, no inversora, donde la señal es acoplada a un nivel de tensión capaz de entrar a los convertidores analógicos digitales. La implementación del circuito se muestra en la figura 3.11.

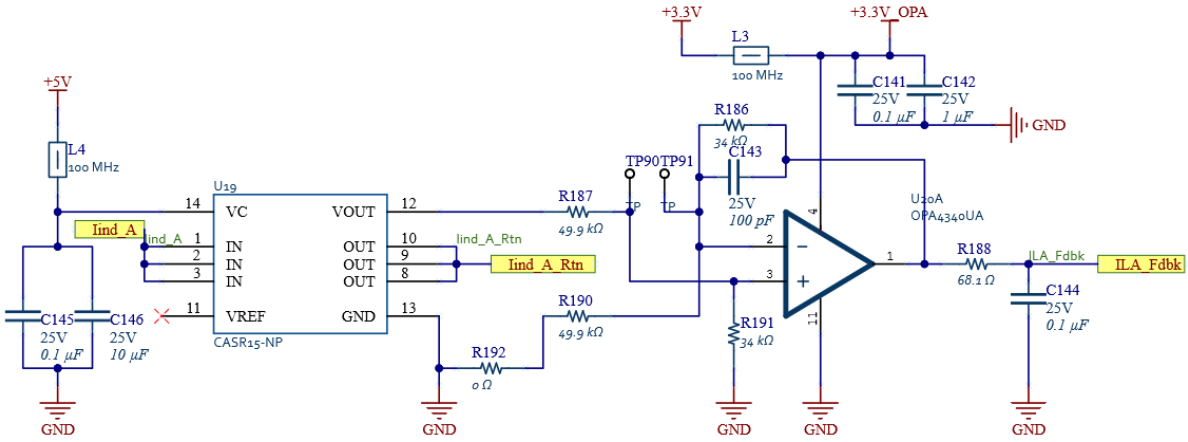


Figura 3.11: Circuito para acoplar la se˜nal de salida del sensor de corriente a la entrada del convertidor anal´ogico digital.

Sensor de Voltaje de Alimentación Para el censado del voltaje de alimentación, es necesario considerar los niveles de voltaje con los que se va a tratar, mismos que fueron definidos anteriormente y rondan en las centenas de voltios de corriente directa. Dicho voltaje debe mantener el aislamiento en el circuito para poder mantener la integridad de los componentes de baja tensión en todo momento. Para ello se diseña una fuente aislada que alimente el lado de alta tensión del amplificador aislado de *Texas Instruments*, con número de parte AMC1311. Donde entrará la línea de alto voltaje, atenuada por 6 resistencias de $332k\Omega$, en una configuración diferencial. El fabricante recomienda poner un capacitor y un diodo *TVS* (*Transient-Voltage-Suppression*) para suprimir los picos de tensión que se pudieran llegar a presentar y así no dañar el amplificador. Para el diseño de la fuente se usa otra parte de *Texas Instruments*, con número de parte SN6505. Dicha parte es un controlador de transformadores para fuentes aisladas de bajo ruido. Esto generará los pulsos necesarios para que el transformador de pulsos empiece a funcionar. A la salida del transformador, se tendrá una configuración de diodos rectificadores conectados al *tap* central y un filtro, dando una salida de aproximadamente 6V que será regulado por un convertidor lineal (TLV70433DBVT), y finalmente se tendrá un voltaje de poco ruido y aislado con valor de $3.3V$. Para el área de baja tensión, se utiliza un interruptor de carga con corriente controlada, TPS22944DCKR. Con el objetivo de garantizar el voltaje de $3.3V$ del amplificador operacional y proteger el circuito regulando la corriente. Se pone una resistencia de sangrado, para evitar la acumulación de carga en las líneas de ambos amplificadores. Finalmente, el segundo amplificador se utiliza en configuración diferencial, usando la salida del mismo tipo del primer amplificador. Acoplando la señal para que pueda entrar a un convertidor analógico digital. La implementación del circuito se muestra en la

figura 3.12 [24].

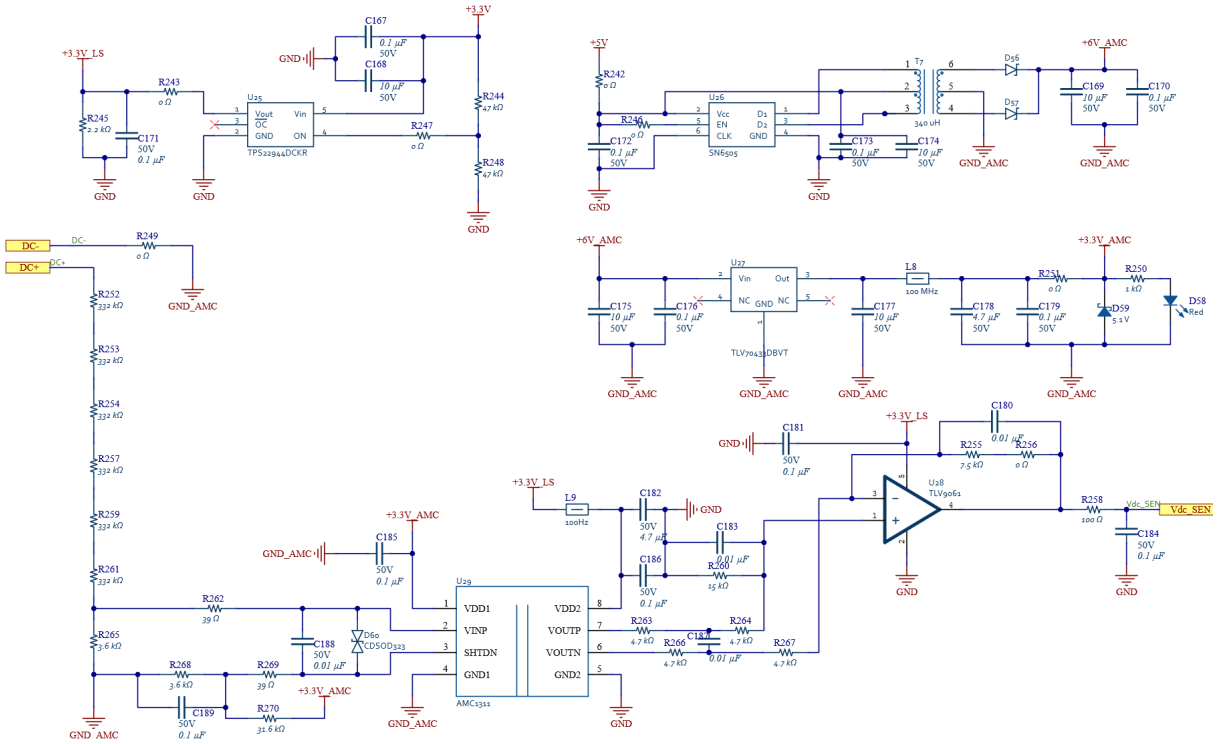


Figura 3.12: Circuito de acoplamiento para una medici´on aislada de voltaje de la l´ınea de entrada, hacia el convertidor anal´ogico digital.

Sensores de Voltaje de Fase Las mediciones de voltaje de cada fase usan otro método de aislamiento que consta de varias resistencias del rango de $M\Omega$ en serie. Para este caso, se ocupan 3 resistencias de $1M\Omega$ en una configuración diferencial no inversora, entre la tierra de potencia del controlador y el voltaje de la fase. Cabe mencionar, que conforme aumenta el voltaje, la señal de salida que va a leer el amplificador va a ser menor conforme ésta aumente. La señal es amplificada en un rango en el que puede ser acoplada a la entrada del convertidor analógico digital. Esta etapa es replicada para cada fase del sistema.

Sensores de Temperatura y Ventilación Como medida preventiva, se decidió poner sensores de temperatura cerca de cada fase para poder apagar el sistema en caso de un sobrecalentamiento. Se dejó una terminal para poder conectar un *NTC* (*Negative Temperature Coefficient*) cerca de cada *MOSFET* y poder tener una especie de fusible térmico. El sensor de temperatura que se utilizó tiene el número de parte LMT87LPM, de la marca *Texas Instruments*. La implementación del circuito se muestra en la figura 3.14.

Como protección, dada la aplicación del circuito, es esperado que el sistema se caliente. Especialmente los transistores de carburo de silicio, mismos que estarán pegados a un disipador que será enfriado por ventiladores. Estos últimos serán controlados por un circuito sencillo de tipo

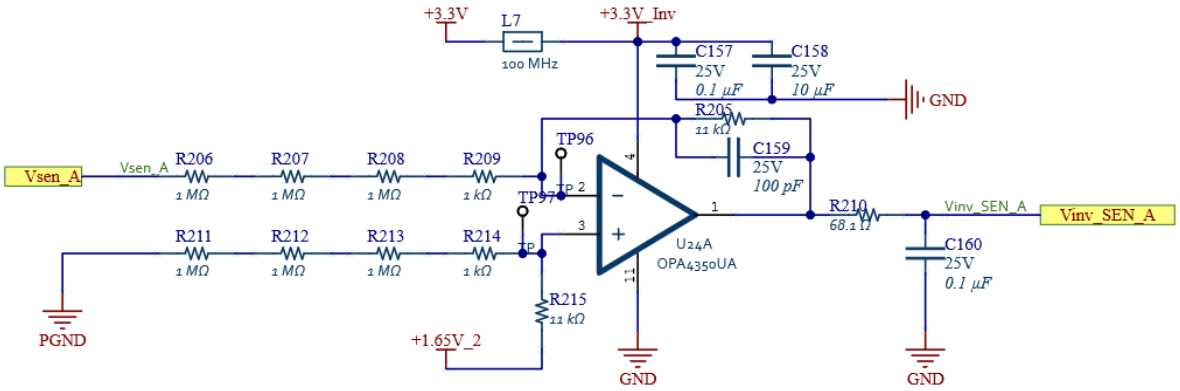


Figura 3.13: Circuito de acoplamiento para una medici´on aislada de voltaje de una fase de salida, hacia el convertidor anal´ogico digital.

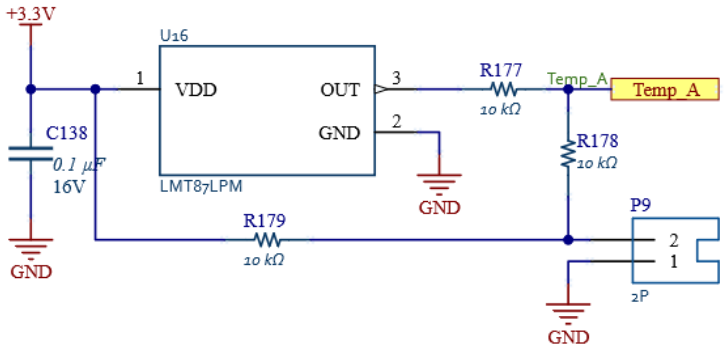


Figura 3.14: Circuito de acoplamiento para la una medici´on de temperatura sobre los conmutadores de cada fase, hacia el convertidor anal´ogico digital.

ON-OFF. La circuitería está planeada para tener un ventilador por fase. La implementación del circuito de ventilación se muestra en la figura 3.15.

Sensor de velocidad El sensor de velocidad utilizado en este sistema consta de un *encoder* rotatorio incremental, cuyo modelo es el LPD3806-600BM-G5-24C que consta de 600 pulsos por vuelta y soporta una velocidad máxima de $2000rpm$. El sensor envía dos señales con pulsos, con las cuales se puede saber la velocidad o posición del motor, así como la dirección de giro. Para la aplicación, sólo se utilizará una línea de pulsos, dado que la dirección de giro es irrelevante. Dentro del *FPGA* se añade una etapa de lectura de pulsos en intervalos de tiempo, para también poder presentar la velocidad del motor en la interfaz con un periodo de muestreo de $1kHz$.

Sensor de Torque El sensor de torque o par, es un dispositivo que mide dicha magnitud en un eje y genera una señal de voltaje proporcional al par ejercido. En esta aplicación, el par medido será de tipo dinámico, por lo que la frecuencia del par debe ser menor a la frecuencia de resonancia

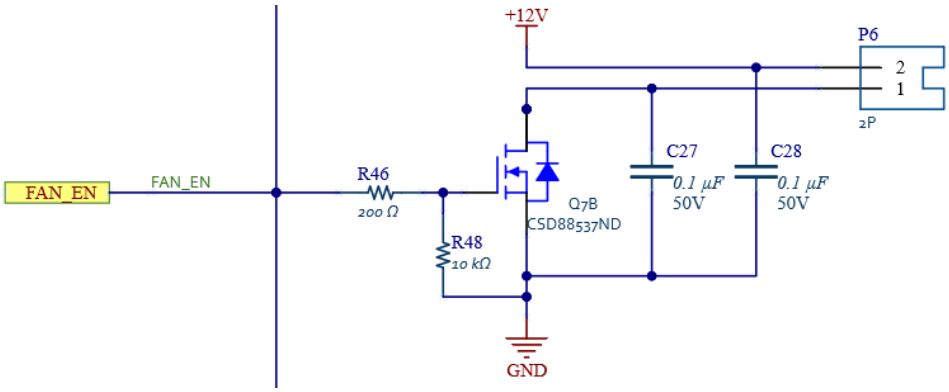


Figura 3.15: Circuito para activar una carga inductiva a trav´es de una entrada salida de la tarjeta reconfigurable.

del conjunto medido. Para tener una medida sin ruido, es necesario colocar el sensor buscando minimizar las vibraciones, así como los cambios de temperatura y, de ser posible, las perturbaciones electromagnéticas. Para este banco de pruebas, de utilizó el sensor de torque con el número de parte FSH01987 de la marca FUTEK. Dicho sensor tiene una capacidad máxima de medición de $10N - m$ y está configurado para ser utilizado como resistencia en un puente de *Wheatstone* como una resistencia de 350Ω, a través de la cual se generarán voltajes muy pequeños (en el orden de los milivoltios) mismos que deberán ser amplificados por una etapa posterior. Para dicha amplificación fue necesario un módulo externo que generaba la compensación y amplificación de la señal.

Convertidor Analógico Digital Los convertidores Analógico Digital son la base de la etapa de monitoreo. Para esta aplicación, se seleccionaron componentes de la marca Analog Devices, con número de parte AD7386. Estos convertidores tienen una resolución de hasta a 14-bits y cuentan con la capacidad de realizar mediciones paralelas en dos de sus cuatro canales a una velocidad máxima de hasta 2MSPS, o el doble de muestras para un canal sencillo. La alimentación del circuito de referencia usa reguladores de precisión. En cuanto a la interfaz, la parte cuenta con un canal dual de comunicación *SPI*, capaz de soportar velocidades de hasta $80MHz$ (no recomendado por implicaciones de la integridad de la señal). El circuito está protegido con filtros RC en las entradas de cada uno de los canales. Cuenta con resistencias y capacitores sobre las líneas de *SPI* en caso que se quiera usar a altas velocidades, con el objetivo de amortiguar el sobrepaso de la señal. La implementación del circuito del convertidor analógico digital, se muestra en la figura 3.16.

3.2.2 Etapa de *VLSI*/Linux

Cuando un *FPGA* modelo *Cyclone V*, de la marca Intel, y un procesador, de arquitectura *ARM*, en este caso un *CORTEX A9*, se encuentran en un mismo bloque de silicio. Forman un sistema en el *chip* (*SoC*), donde las capacidades de comunicación entre ambas partes se ve maximizado. Ya que existen varios hilos que se conectan directamente a la línea de comunicación del procesador.

Sin embargo, para poder crear una interfaz entre el uso de un sistema operativo con un hardware ser modificado según la descripción que se realice en la lógica del *FPGA*, complica la tarea y hay que investigar a muy bajo nivel lo que sucede para poder hacerlo de forma correcta. Ahora, basándose

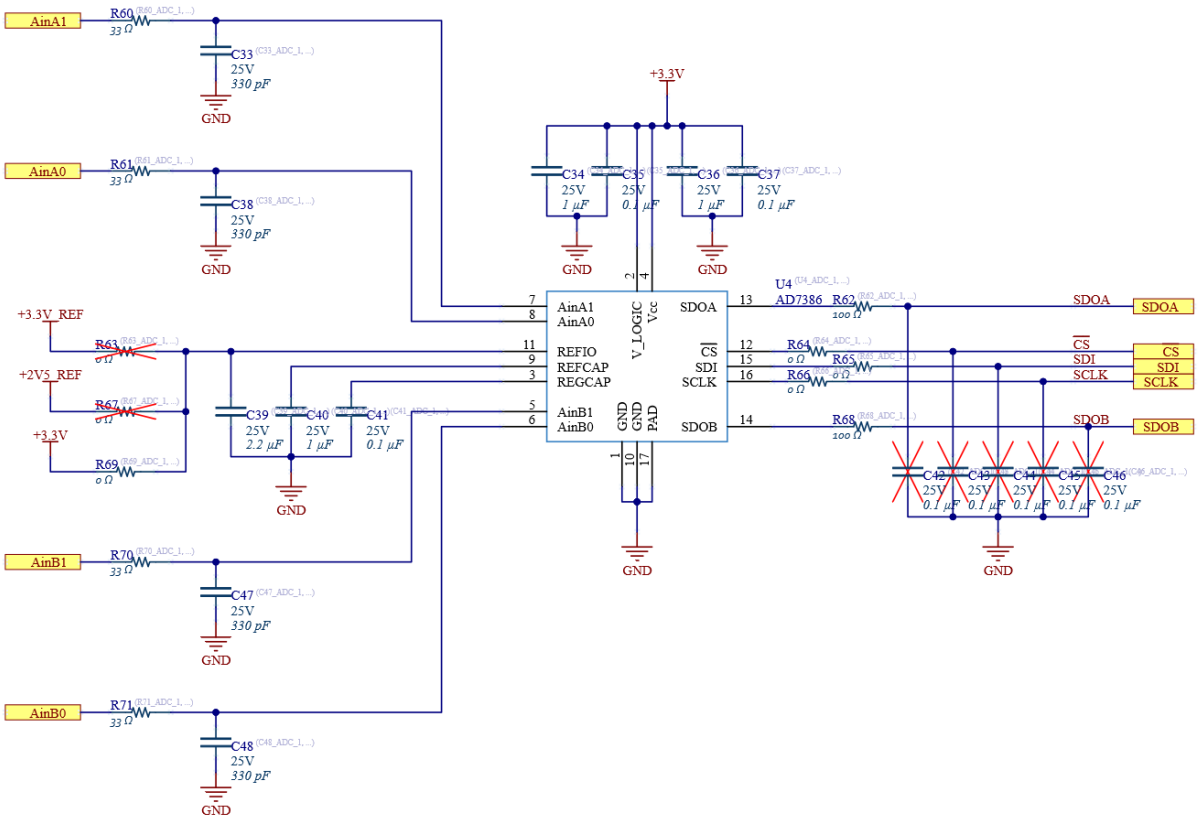


Figura 3.16: Circuito b´asico del convertidor anal´ogico digital con fuente de referencia externa y filtro pasivo de entrada.

en el circuito descrito en la sección anterior y tomando en cuenta el diagrama de bloques inicial, se procede a definir el diagrama de entradas y salidas para la circuitería descrita a lo largo de la sección como se muestra en la figura 3.17.

Para poder implementar el sistema de forma correcta se debe seguir el proceso de desarrollo para sistemas embebidos que utilizan como base el sistema operativo Linux, con sus respectivas variaciones para poder integrar el *FPGA*.
Primero se deberá crear una configuración para el *FPGA*. La cual debe incluir los bloques para habilitar los periféricos de la tarjeta de desarrollo, como lo son controladores de Ethernet, *DDR3*, *Buses AXI*, Memorias *SD/MMC*, *USB*, *UART*, *I2C*. Posteriormente, los bloques secundarios como controladores de *CAN*, *SPI*, *GPIOs*, *FIFOs*, y los bloques de control y monitoreo diseñados para la aplicación. Para esto, se utilizará el *software Prime* versión 20.1.
Una vez que la configuración del *FPGA* está lista, se procede a utilizar el programa de Intel *Embedded Development Suite (EDS)* que generará el *Bootloader* primario y lo compilara. Lo siguiente, es crear un *Bootloader* secundario con la herramienta u-boot, que se encargará de cargar la configuración inicial en el *FPGA* e inicializará el sistema operativo, en este caso Linux. La distribución que se utilizara de Linux es una personalizada, que toma como referencia a poky, un sistema mínimo reconfigurable del proyecto Yocto. Misma que será compilada con las herramientas necesarias,

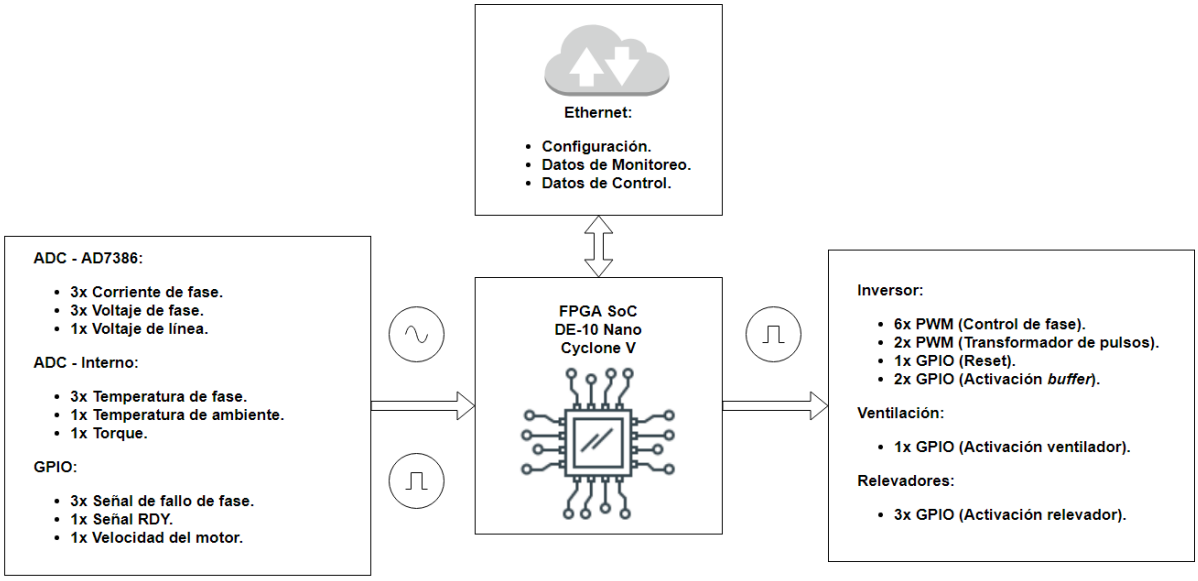


Figura 3.17: Diagrama simplificado de entradas y salidas.

como net-tools y python 3 que se van a utilizar para crear una interfaz entre la tarjeta a la red y a mandar datos al usuario. Por lo que el agregar dichas herramientas al sistema operativo será fundamental para los siguientes pasos.
El *Device Tree* es la parte donde se le dice al kernel con qué periféricos se cuentan. Aquí se describirán esas interfaces como *UART*, *SPI*, *I2C* y *CAN* para que podamos hacer funciones que interactúen con señales eléctricas. Finalmente, se integrarán los archivos generados en las etapas previamente descritas en una imagen que será cargada a una memoria para que el procesador pueda comenzar a operar.

Diseño *RTL* (*Register-Transfer Level*)

El ciclo de desarrollo para el diseño digital parte de la creación de bloques que se interconectan para generar sistemas complejos. Los bloques serán utilizados para crear los módulos de adquisición de datos y para interactuar con las otras etapas del inversor. Estos módulos parten de bloques más básicos como *latches*, contadores, temporizadores, serializadores, entre otros. Los bloques anteriores fueron descritos y validados en el lenguaje de descripción de hardware llamado Verilog, utilizando herramientas *open source* como Icarus Verilog. Al unir bloques se crearon módulos, con funcionalidades más complejas como memorias *FIFO*, transductores de *SPI* dual para los convertidores analógico-digital, sistemas de detección de fallas, y el controlador de motor.

Para la integración de cualquier configuración del *FPGA* con el microprocesador, se debe considerar que el *FPGA* realiza la conexión entre el microprocesador y la memoria RAM externa, por lo que debe existir un módulo base con dichas condiciones para poder ejecutar un sistema operativo en la tarjeta. Para ello, existen soluciones que son provistas por el fabricante, Intel, que facilitan la implementación del proceso anteriormente mencionado. La herramienta *"Platform Designer"* es la base para interconectar de una forma amigable con el usuario todos los módulos que se diseñaron y validaron. Nuevamente, partiendo de un diseño modular que puede ser fácilmente interconectado,

siguiendo los protocolos de comunicación del *FPGA(Avalon)* y del microprocesador (*AXI*).

Bootloader Primario, *Preloader*, U-boot

El *bootloader* primario es generado automáticamente por la aplicación *EDS*, provista por Intel. Este *bootloader* primario es el encargado de la inicialización mínima de los periféricos (puerto serial, interfaces de memorias, interfaces de comunicación) y copiará la información necesaria para ejecutar el *preloader* a la memoria RAM dentro del circuito integrado. Asimismo, lo ejecutará. El *preloader* va a configurar los relojes, entradas y salidas, configuraciones de multiplexor, así como inicializar la memoria RAM externa donde copiará los archivos del *bootloader* secundario para así ejecutarlo. U-boot es el encargado para crear el *bootloader* secundario, que es más complejo que el anterior. Sin embargo, para fines prácticos, se simplifica en que es el encargado de configurar inicialmente al *FPGA* y de cargar el sistema operativo a la memoria RAM para poder ejecutarlo.

Yocto - Sistema Operativo Linux

Yocto es una comunidad que ayuda a los desarrolladores a hacer una distribución personalizada de Linux, sin importar la arquitectura del hardware donde se vaya a correr el sistema operativo. Entenderlo es complicado, ya que se debe tener un conocimiento sobre los componentes necesarios para inicializar un sistema operativo desde el nivel eléctrico y por las capas más bajas de abstracción. Mismas que se explicaron en secciones anteriores.

Para poder compilar Linux se usa un sistema operativo de referencia provisto por Yocto, de nombre poky. Este sistema operativo cuenta con lo más básico para correr Linux, de forma que una imagen mínima puede pesar desde 2.2MB. Cabe mencionar que en ese punto el sistema no posee nada, por lo que se deben añadir elementos, que en el entorno de programación de Yocto se le conoce como recetas. Las recetas contienen las herramientas que se utilizaran para las aplicaciones del sistema operativo, así como la descripción de los elementos internos de la arquitectura del procesador que se esté utilizando, conocidos como *Board Support Platform* (*BSP*). El BSP de las tarjetas comerciales como RPI, BeagleBone, y el DE-10 Nano son provistos por el proveedor, en este caso Altera, hoy en día Intel.

Para compilar nuestro sistema operativo personalizado usando como referencia poky y utilizando las herramientas del proyecto Yocto, basadas en Python, como bitbake, se recomienda utilizar el sistema operativo CentOS 7 o Ubuntu 18.0. Debido a que el compilador para esta versión (Dunfell) debe ser gcc versión 9.3, al momento de la escritura de esta tesis. Se hace énfasis en este paso, dado que el intento inicial para compilar poky se hizo con Arch Linux, usando gcc versión 11.1.0 y existían errores en la misma por la versión del compilador. En la figura 3.18 se muestra como verificar la versión del compilador GCC.

```
titan@titan  tutorial gcc --version
gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Figura 3.18: Revisi´on de versi´on del compilador GCC.

Habiendo validado este punto, se procede a clonar el repositorio de poky con el siguiente comando:

```
git clone git://git.yoctoproject.org/poky.git
```

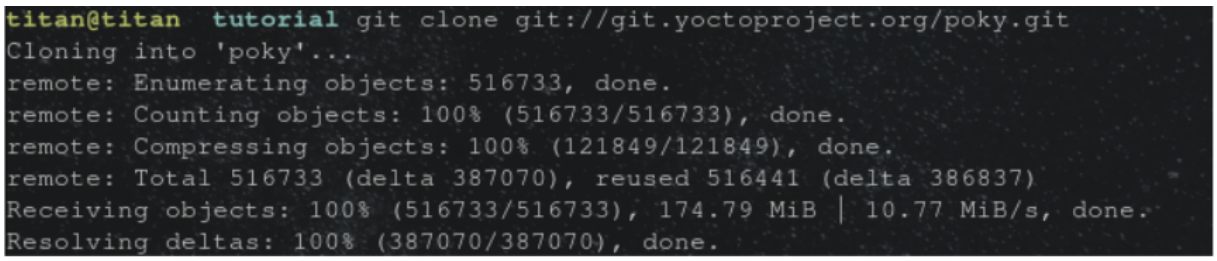


Figura 3.19: Respuesta tras clonar el repositorio de poky a nuestra computadora.

Una vez dentro de la carpeta, se debe seleccionar la versión de poky que se pretende usar, en este caso será la versión Dunfell. Para lo cual se corre el siguiente comando:

```
git checkout dunfell
```

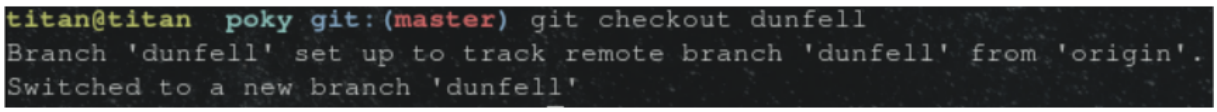


Figura 3.20: Selecci´on de versi´on de poky a utilizar como sistema operativo de referencia.

Ahora se debe inicializar el entorno donde se pretende configurar y compilar el sistema operativo y donde se van a adjuntar las recetas con las herramientas para cargarlo en la tarjeta. Para ello se usa el siguiente comando:

```
source oe-init-build-env
```

A su vez, este comando crea una carpeta con el nombre “build” donde se guardan los archivos de configuración y los archivos generados al exportar el sistema operativo. Dentro de esta carpeta se encuentra otra carpeta con el nombre “conf”, en ella hay dos archivos clave para especificar el objetivo y las herramientas que se implementarán en la compilación. Para este caso, al querer hacer la implementación en una tarjeta de Intel, se debe descargar su receta de BSP. Misma que incluye la información de la arquitectura para hacer la compilación cruzada de forma correcta. La receta será copiada en la carpeta “poky” desde git, con el siguiente comando.

```
git clone git://github.com/robseb/meta-intelfpga.git
```

La filosofía de Yocto se basa en un sistema modular, donde se van agregando elementos a la base del sistema operativo, de modo que es muy fácil detectar cuando una implementación es incorrecta. A estos módulos que se añaden a la receta se les conoce como meta capas y se pone el prefijo "meta-" antes del nombre. Además de contener la información de *BSP*, y hace referencia a una meta capa de nombre "meta-openembedded" que permitirá a bitbake instalar todas las dependencias para poder ejecutar *python* y pip, así como herramientas de red, entre otras. Finalmente, antes de ejecutar el último comando se verifica que en la carpeta de configuración el archivo bblayers.conf añada la meta capa mencionada anteriormente y que en el archivo local.conf, se seleccione como máquina objetivo "cyclone5". Esto garantizará que al generar la distribución de Linux, contaremos con todas esas herramientas y no se necesitará instalar nada más. Entonces, se ejecuta.

```
bitbake core-image-minimal
```

Kernel - *Device Tree*

El *device tree* es un archivo auto generado por las herramientas de Intel, donde se establecen todos los detalles del procesador donde se va a ejecutar el sistema operativo. En este documento se describen las direcciones de memoria de los periféricos como tren de datos, relojes, *watchdogs*, memorias, protocolos de comunicación, salidas y entradas de propósito general, accesos directos a memoria. Igualmente, los núcleos del procesador y la memoria caché. Este archivo tiene terminación ".dts" y es compilado a través de una computadora con Linux.

Memoria SD - Build System

La memoria de la cual se copiará toda la información descrita anteriormente, debe contar con una estructura predeterminada, de lo contrario no funcionará. La memoria deberá contar con tres particiones. La primera partición es de tipo "raw" y contendrá una imagen precompilada del *bootloader*. La segunda partición es de tipo "ext_3" y contendrá la estructura de archivos de Linux que fue generado por *bitbake (Yocto)*. La tercera partición es de tipo "fat" y contendrá una imagen precompilada del kernel de Linux y archivos como el *device tree*, la configuración del *FPGA* y cualquier documento que se vaya a incluir al sistema operativo.

Módulos de Kernel

El fabricante de estos dispositivos provee al cliente una serie de librerías que hacen llamadas al sistema para poder tener un intercambio de información entre el *FPGA* y el procesador ejecutando Linux. Estas librerías son archivos en lenguaje C y facilitan una serie de comandos con los cuales el usuario puede generar funciones para mandar y recibir datos a través de las dos líneas de transmisión que existen entre el *FPGA* y el microprocesador (32-bits y 128-bits). Usando estas mismas librerías, el usuario puede reconfigurar el *FPGA* en tiempo real, cargando archivos binarios que son generados por Quartus. El principio de funcionamiento radica en asignar al bus, que conecta el *FPGA* con el procesador, una dirección de memoria. Misma dirección será apuntada al momento de hacer cualquier tipo de intercambio de información con el *FPGA*. Van a existir 3 direcciones principales, dependiendo de el bus que se ocupe.

Estos módulos de kernel serán utilizados para enviar comandos a la *FPGA*, para recibir las lecturas del *ADC* y para programar el *FPGA* en tiempo real.

3.2.3 Interfaz

La interfaz gráfica se programa con el objetivo de presentar los datos y el panel de control para accionar el motor. Dicha herramienta es programada en el idioma de programación *python*. Dentro del cual se utilizan librarías como PyQt, que facilitan el desarrollo de un software gráfico; *sockets*, que permiten la comunicación dentro de la red para acceder al *FPGA SoC*; mathplotlib, que facilita la implementación de gráficas en la interfaz. La estructura se basa en tener los botones y parámetros del panel de control del motor del lado derecho y los controles para ajustar las variables a graficar del lado izquierdo. Las gráficas se abren en ventanas independientes, para poder tener visibilidad de múltiples gráficas (máximo 4). En la parte del (*backend*), se tiene un servidor corriendo que esta en constante comunicación con el dispositivo en campo. Dicha comunicación se logra por medio de la implementación de *sockets*, que son una *API* (*Application Programming Interface*) utilizada para enviar mensajes a través de la red. Sin embargo, para poder implementar *sockets* de manera satisfactoria, es necesario tener por lo menos dos nodos, donde uno tome la postura de servidor y el otro como cliente. Por dicho método se reciben los paquetes con la información que se va a graficar y se envían las órdenes de control del motor.

Capítulo 4

Resultados

4.1 Resultados

Para llegar a este punto del proyecto, tuvieron que pasar varias iteraciones del diseño del hardware. Los motivos principales fueron, por un lado, el mal diseño de algunas tarjetas y por otro lado, la curiosidad de aprender nuevos conceptos para ponerlos en práctica y utilizar las tecnologías más nuevas e innovadoras para lograr un sistema poco convencional.

Se puede decir ahora que, efectivamente, es posible construir una estación de pruebas y moniotreo que es capaz de operar y visualizar variables de motores de corriente alterna trifásicos, que esta basada en un sistema embebido de alta velocidad y cuenta con una interfaz en tiempo real.

Dada la magnitud de las múltiples etapas con las que cuenta el sistema, los resultados serán divididos en tres partes. La parte de hardware, cubriendo los detalles y retos que se tuvieron al momento de realizar la tarjeta final; la parte de software, cubriendo los aspectos de programación tanto del *FPGA*, como del sistema operativo y la interfaz; la parte de el resultado final del sistema integrando todos sus elementos.

4.1.1 Resultados - Hardware

En total, a lo largo de esta tesis se diseñaron y fabricaron cuatro versiones de inversores y dos versiones de tarjetas de muestreo. Y es evidente que existe una diferencia significativa entre el primer y el último diseño de las tarjetas.

El primer inversor, estaba planeado para utilizar un Arduino Nano que controlara el motor. El circuito estaba optoacoplado, para aislar las señales de entrada de los impulsores y contaba con tres pares de *MOSFETs* que estaban mal colocados, a los cuales no se les podía poner un disipador. Los componentes en este diseño eran de tipo *thru hole*. El modelo 3D de este circuito se muestra en la figura 4.1.

El segundo inversor, fue el mismo circuito anterior, pero el Arduino Nano fue reemplazado por el mismo procesador (ATMEGA328) y los componentes utilizados fueron reemplazados por componentes de montaje superficial. La parte de potencia quedó igual, con la diferencia que se le puso un capacitor electrolítico y uno cerámico en la entrada para filtrar cualquier tipo de ruido en el voltaje de entrada. El acomodo de los *MOSFETs* no fue corregido, por lo que no era posible ponerle un disipador al sistema. El modelo 3D de este circuito se muestra en la figura 4.2.

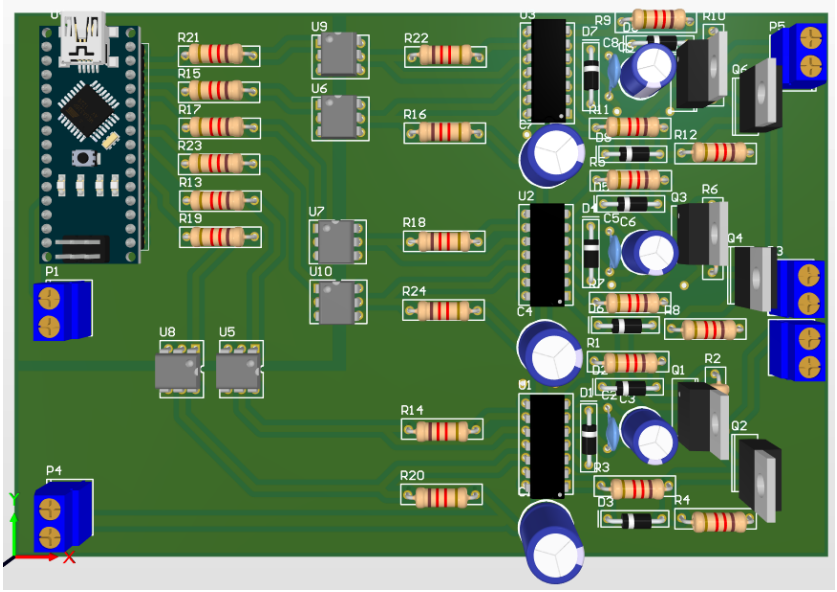


Figura 4.1: Primera iteraci´on del inversor trif´asico.

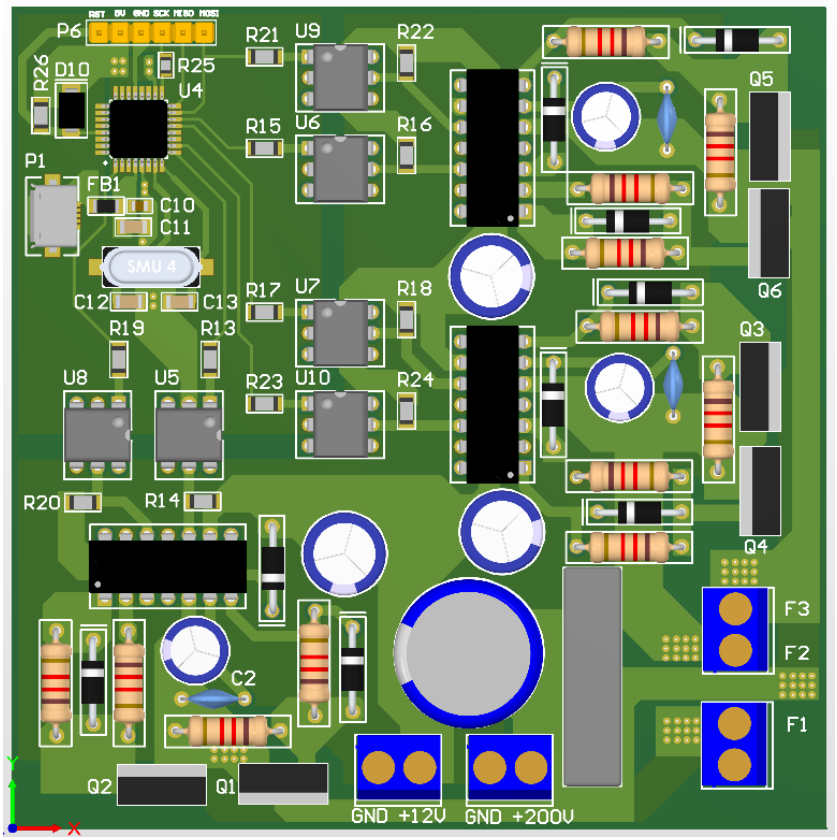


Figura 4.2: Segunda iteraci´on del inversor trif´asico.

El tercer inversor, fue un avance significativo para el proyecto ya que incrementaba la complejidad del esquemático, así como la interconexión de la tarjeta. Para este diseño, los impulsores y optoacopladores cambiaron a ser de tipo montaje superficial, al igual que todos los componentes de la tarjeta, a excepción de los *MOSFETs* que fueron sustituidos por un empaquetado más grande y con mejores propiedades para disipar potencia. La arquitectura era la misma, a diferencia que ahora, se contaba con sensores de corriente para cada fase. Y, la parte de procesamiento digital, se haría en otra tarjeta. En este momento se migró el proyecto de un microcontrolador de 8 bits a un microcontrolador de 32 bits de la marca ST, con arquitectura de procesador dual Cortex M7 y Cortex M4, que contaba con una pantalla táctil integrada a la tarjeta de desarrollo. Sin embargo, esta tarjeta no fue la solución ya que carecía de protecciones y varios impulsores fueron quemados durante las pruebas para implementar los algoritmos de conmutación de forma correcta. Después de diez impulsores quemados, era momento de hacer algo más robusto. Este diseño fue interconectado utilizando un *PCB* de 4 capas, y el conector de la tarjeta estaba planeado para acoplarse con la tarjeta de desarrollo de ST Microelectronics, con número de parte STM32H745I-DISCO. La tarjeta del tercer inversor se muestra en la figura 4.3.

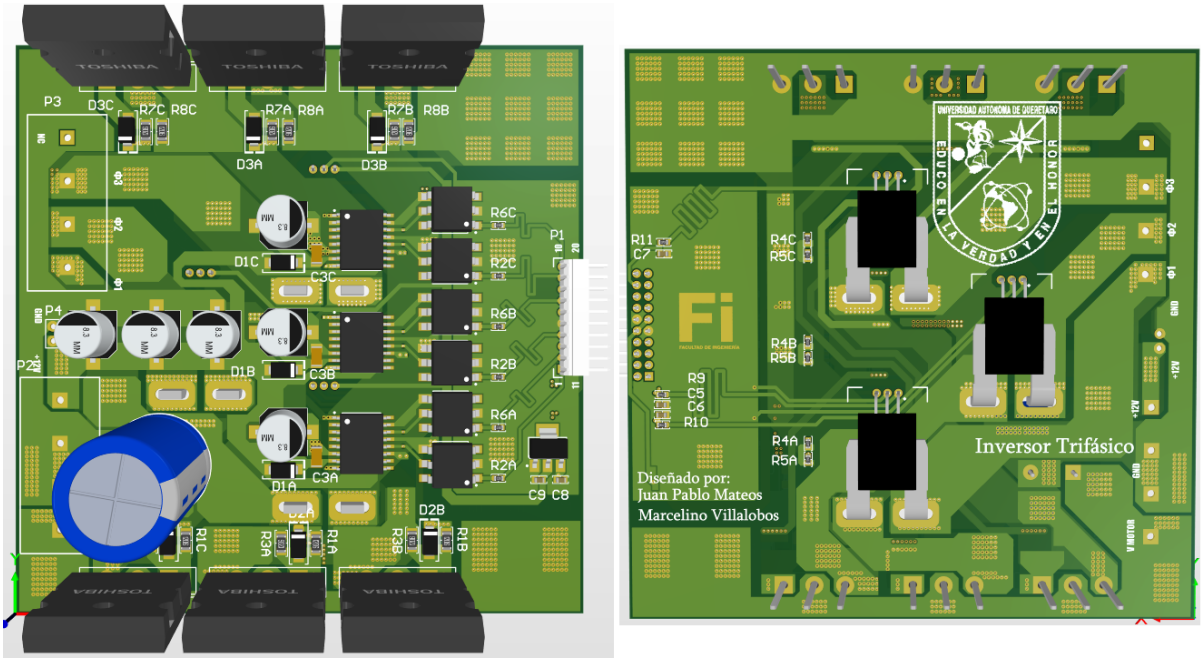


Figura 4.3: Tercera iteraci´on del inversor trif´asico. Del lado izquierdo la vista superior, y del lado derecho la vista inferior.

La primera tarjeta de muestreo, se diseñó junto con el segundo inversor. Esta tarjeta tenía como entradas los sensores de voltaje, corriente y velocidad del motor y hacía el acoplamiento para los convertidores analógicos digitales del ATMEGA328. Esta tarjeta necesitaba de dos fuentes externas, un voltaje positivo y uno negativo para el correcto funcionamiento de los amplificadores operacionales. Sin embargo, al utilizar los convertidores del micro controlador, no se podían realizar mediciones paralelas sobre las señales de corriente o voltaje. Por lo que se optó por realizar otra tarjeta de muestreo con un acercamiento distinto. La primera tarjeta de muestreo se muestra en la

figura 4.4.

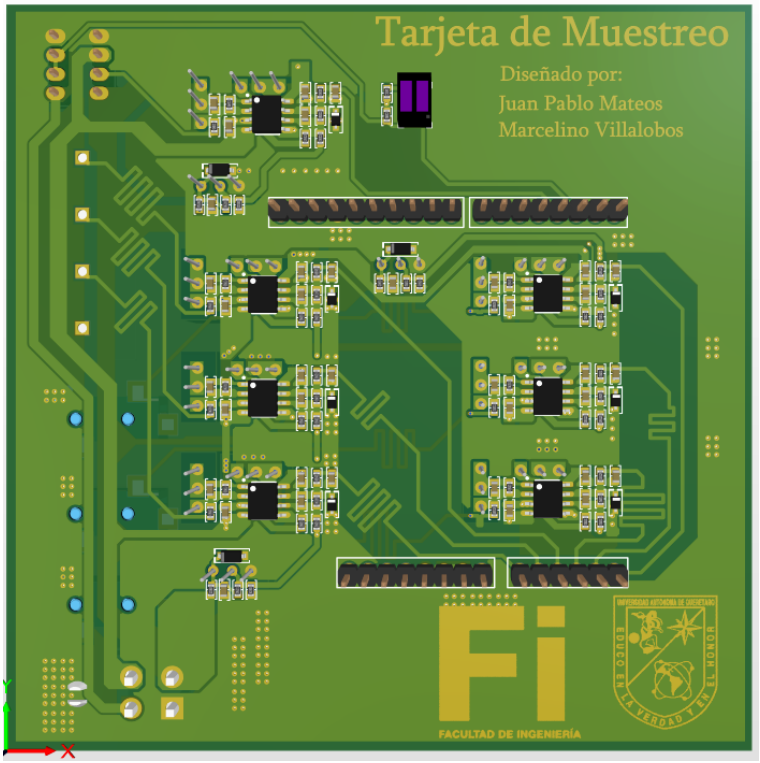


Figura 4.4: Primera iteraci´on de la tarjeta de muestreo.

Buscando solucionar el problema de las mediciones paralelas, surge la segunda versión de la tarjeta de muestreo. La cual integra, no sólo la etapa de acoplamiento de señales, si no que también cuenta con tres convertidores analógico digital de alta velocidad (AD7386) que permiten muestrear paralelamente dos canales cada uno. Además cuenta con fuentes de voltaje de bajo ruido, así como circuitos integrados de referencia de voltaje, para administrar el voltaje tanto negativo como positivo de los amplificadores operacionales con sólo una alimentación a la tarjeta de 5V. Para el voltaje negativo, se utilizó un convertidor de tipo *charge pump*. Para el momento del diseño de esta tarjeta, ya era necesario utilizar un *FPGA* que pudiera proveer a los convertidores seis líneas de comunicación *SPI* de forma paralela. Esta tarjeta superó las expectativas, ya que proveía mediciones muy precisas y alcanzaba una velocidad de muestreo de 1 millón de muestras sobre segundo con una resolución de 14 bits. En términos de integridad de señales para las líneas del protocolo *SPI*, existía la duda de la velocidad máxima de transmisión que se podría llegar con el *FPGA*. Ya que dentro del camino de las señales, el conector representaba una variación en la impedancia de las pistas que iba a generar un rebote de la señal en la línea de transmisión, que podría llegar a causar una distorsión. Dicho problema no iba a ser analizado a detalle, ya que sale de los alcances del proyecto, sin embargo era un posible modo de fallo en el sistema. Al implementar el protocolo *SPI* a $50MHz$ de forma paralela a los tres convertidores, no hubo ningún problema y la medición se obtuvo sin mayor problema. La segunda tarjeta de medición se muestra a continuación en la figura 4.5

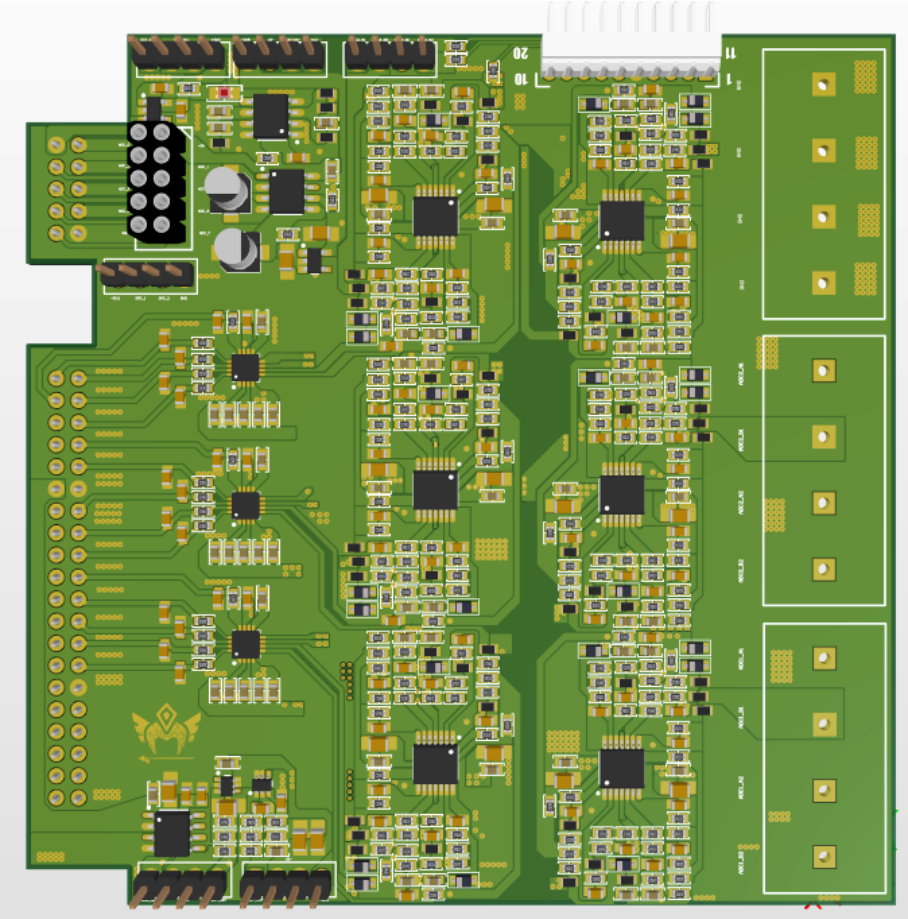


Figura 4.5: Segunda iteraci´on de la tarjeta de muestreo.

Finalmente, al aunar todos los conocimientos y lecciones aprendidas de los circuitos anteriores, se procedió a diseñar un cuarto inversor. El cual tenía una arquitectura mucho más compleja a los anteriores y fue basada en un modelo de referencia de *Texas Instruments* que no estaba a la venta. Además, a la misma tarjeta del inversor se le incorporaron los convertidores de la segunda tarjeta de muestreo, dando como resultado una tarjeta única, capaz de soportar más potencia que las anteriores ya que los *FETs* utilizados en este proyecto son con la tecnología de carburo de silicio. Esta tecnología, nos obligaba a tener que poner fuentes aisladas de +20V y -5V, para garantizar las transiciones de prendido y apagado de dichos *FETs*. Y a cambio, su velocidad de conmutación en altos voltajes es superior a la de *MOSFETs* convencionales. El diseño cuenta con fuentes de alimentación para que sólo se tenga que alimentar con una fuente de entre 12V y 16V. Posee un par de *buffers* para proteger los pines del *FPGA*, al igual que sensores de corriente y voltaje con sus respectivas etapas de acoplamiento que terminan en un par de convertidores analógico digital. A esta tarjeta se le añaden sensores de temperatura para poder accionar ventiladores en caso de ser necesario disipar el calor de forma más rápida. También cuenta con una serie de conectores para comunicarse con los sensores de velocidad y torque, que se encuentran externos al circuito, pegados al motor. Para aislar el área de potencia, de el área digital, se utilizan transformadores

de pulsos que son accionados por el *FPGA*, a través de una configuración de medio puente H, que conmuta a $500kHz$. Esto genera una señal de salida de aproximadamente 25V, misma que será dividida con un diodo Zener ajustable, para que se genere una tierra flotada y se obtengan los +20V y los -5V. La etapa antes mencionada tuvo que ser modificada, añadiendo un regulador variable, puesto que el voltaje de salida era mayor a +25V y tuvo que ser ajustado. Ese voltaje de salida aislado es rectificado con una configuración de *tap* central, y posteriormente es filtrado, ya que los transformadores de pulsos son conocidos por insertar muchos componentes de alta frecuencia en las líneas de salida. En el lado de potencia se encuentran dos impulsores por fase, mismos que van a controlar los *FETs*. Estos impulsores cuentan con funcionalidades de protección para altas corrientes, a través de un comparador. Una de las lecciones aprendidas, que costó más tiempo diagnosticar fue en esta etapa. El uso de componentes de baja calidad afecta el comportamiento final del circuito, en este caso, un diodo Zener estaba conduciendo picos de 9V cuando era polarizado de forma directa y su voltaje nominal, cuando era polarizado de forma inversa. Para solucionar este problema, se soldó diodo en serie, que bloqueara los picos de voltaje, cuando el Zener se polarizaba de forma directa. De esa forma, se mitigaron las activaciones erróneas del circuito de protección. Finalmente, como protección adicional, se pusieron relevadores en serie con la salida de voltaje para evitar cualquier tipo de descarga mientras el motor no estuviera conectado. Esta versión final, se creó en un *PCB* de 4 caras y sus dimensiones son 20cm de longitud por 18cm de altura. La tarjeta final se muestra en la figura 4.6.

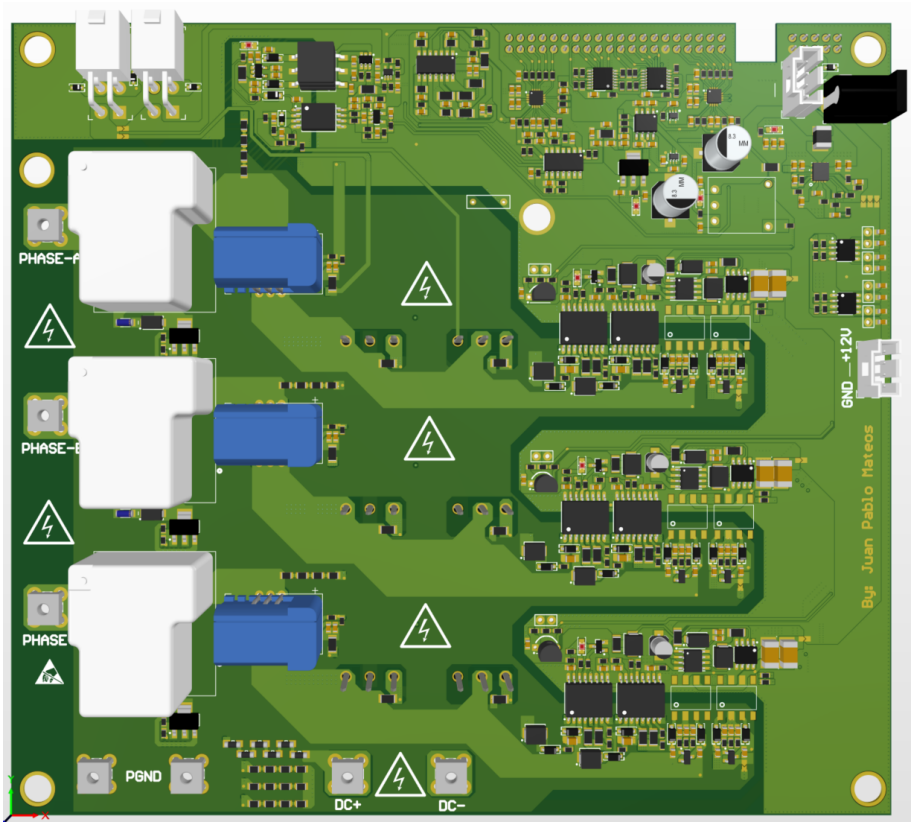


Figura 4.6: Cuarta iteraci´on del inversor trif´asico.

4.1.2 Resultados - Software

El software de este proyecto se dividió en tres etapas diferentes. Dichas etapas fueron el desarrollo de la descripción de hardware del *FPGA*, la creación del sistema operativo capaz de conectarse con el *FPGA* y el desarrollo de la interfaz capaz de compartir información con el sistema.

Hacer un módulo a base de una serie de bloques descritos en algún lenguaje de descripción de hardware, como lo es verilog, y ponerlos a sintetizar para generar un archivo de configuración para un *FPGA*, sin haber validado previamente los bloques internos de forma individual y posteriormente en conjunto, es una maniobra atrevida que la mayoría de las veces termina en una pérdida de tiempo. El motivo por el cual se utilizó Verilog como lenguaje de descripción de hardware, es por la serie de herramientas que cuentan para fácilmente validar los bloques y los módulos. Para cada *latch*, contador, *timer*, transceptor *SPI*, entre otros, se creó un archivo de prueba para validar el funcionamiento de los mismos a través de una herramienta llamada Icarus Verilog y graficando los resultados con GTKWave. A través de estas simulaciones, se inducían señales de prueba a los módulos y se validó el correcto funcionamiento de cada parte. Es importante recalcar que cada módulo se diseñó de forma genérica para poder escalar los tamaños de sus entradas y salidas para poder reutilizarlos lo más posible. Posteriormente, con el apoyo de Quartus y su herramienta *Platform Designer*, fue posible crear un bloque para inicializar la memoria RAM y, al mismo tiempo, interconectar los bloques descritos con el procesador del *FPGA SoC*. Asignando así una dirección de memoria para cada módulo dentro de la línea de transmisión. En la figura 4.7 se muestra la herramienta, del lado izquierdo se muestra la interconexión de los bloques descritos a las líneas de transmisión y del lado derecho se muestran las direcciones asignadas a los módulos. Es en esta etapa donde se abre el espacio de introducir un controlador de lógica difusa que manipule las variables del motor. Al introducir un bloque de dicho controlador, es posible conectarlo al bus principal y poder compartir parámetros con la interfaz gráfica, utilizando los protocolos de comunicación previamente establecidos.

Por otro lado, un sistema tan detallado para la aplicación descrita, necesita de un sistema operativo que le brinde compatibilidad con los estándares de otras máquinas o computadoras para poder comunicarse con ellas. Sin embargo, se buscó minimizar la carga de procesamiento de esta tarjeta, ya que a mayor carga de instrucciones mayor es la potencia que consume el *FPGA SoC*, que se reflejará en un incremento de temperatura. Partiendo de ese punto, fue posible generar una imagen con las herramientas provistas por Yocto, que contuviera solamente la cantidad indispensable de programas necesarios para poder adquirir datos del *FPGA* y mandarlos a través de la red a la computadora del usuario. Dentro de las herramientas destacan las recetas meta-intel que contiene el *BSP* de la tarjeta (todas las librerías para interconectar el procesador a sus periféricos), meta-openembedded que contiene python, pip, net-tools y gcc, con sus respectivas dependencias. Cuando el sistema operativo estuvo listo, se creó un controlador para reconfigurar el *FPGA* a través de Linux, y un par de controladores para leer y escribir información a través de las líneas de transmisión del *FPGA*. Posteriormente, se crearon unos *sockets*, utilizando python, con los cuales se definieron métodos de instrucciones para poder interactuar con el sistema y poder mandar información de forma bidireccional. Los comandos permiten modificar el ciclo de trabajo y la frecuencia de las señales de entrada del motor, así como comandos de inicialización de los módulos de *ADC*, entre otros.

Finalmente, para poder converger la información de una forma gráfica, se diseño una interfaz donde el usuario pudiese interactuar, a través de botones y gráficas, con el motor. Dicha interfaz, implementa los métodos definidos por la etapa pasada para poder generar un flujo de información.

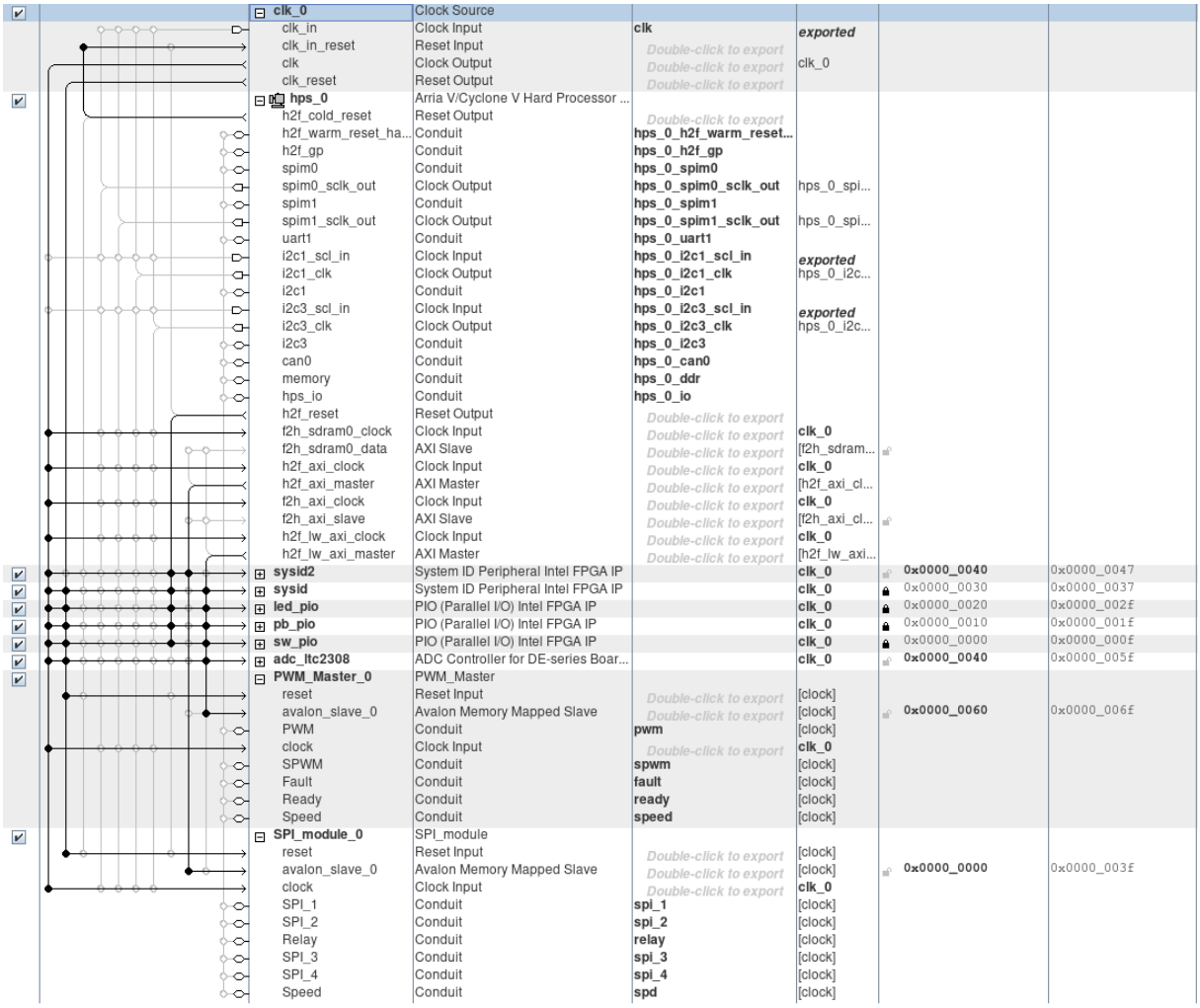


Figura 4.7: *Platform Designer*, una herramienta de Quartus, que simplifica la interconexi´on de m´odulos en el dise˜no de sistemas escalables con protocolos propietarios como *Avalon* y *AXI*.

Lo que permite tener una actualización dinámica de las variables del motor, que pueden ser graficadas en tiempo real. El sistema permite un método de selección de variables a graficar en hasta cuatro gráficas diferentes. Dichas gráficas están compuestas de 30000 muestras por variable, considerando un periodo de muestreo de 5000 muestras por segundo, a las cuales se les aplica un filtro pasa-bajas digital antes de ser graficadas. Por otro lado, muestra un par de campos, donde se define la frecuencia y el ciclo de trabajo de la señal de entrada del motor, así como un par de motores que permiten inicializar o apagar el motor. La interfaz gráfica utilizó una serie de librerías como PyQt5, matplotlib, filtros digitales y *sockets* para facilitar el desarrollo. Cabe mencionar que esta etapa se corre dentro de la computadora del usuario, lo que aligera la carga del *FPGA SoC* y reduce los tiempos de procesamiento, dadas las capacidades de las computadoras personales de hoy en día. El sistema es capaz de actualizar dinámicamente las gráficas, cada que la información recolectada es leída por el procesador y de mandar comandos instantáneamente para accionar el motor. La

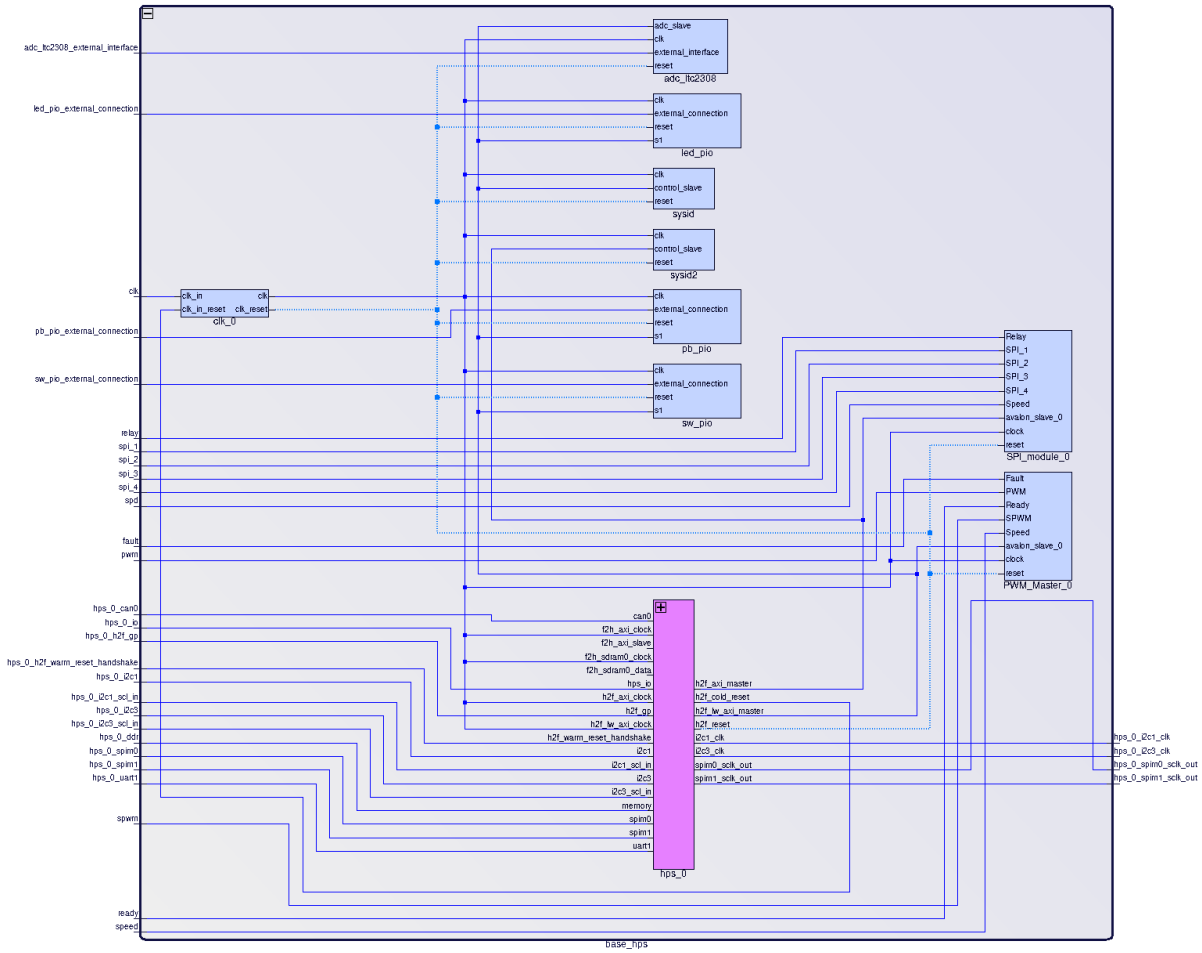


Figura 4.8: Diagrama a bloques, mostrando de forma gr´afica los componentes del sistema y sus respectivas l´ıneas de entrada y salida del sistema.

interfaz se muestra en la figura 4.9.

4.1.3 Resultados - Sistema

Se realizaron pruebas de voltajes y corrientes para validar que cada bloque de la tarjeta del inversor funcionara correctamente. Los voltajes nominales de las fuentes de alimentación eran los esperados y no existían corto circuitos en la tarjeta. En el área de potencia se corroboró que el voltaje de salida de los transformadores de pulsos es el correcto, el circuito es capaz de conmutar las tres fases y que en todos los casos los *FET* se activaban y apagaban con las señales inducidas por el impulsor. Los relevadores conmutaban según las señales de entrada del *FPGA SoC*. Las protecciones de sobre-corriente fueron activadas de forma satisfactoria al cortocircuitar alguna fase. En el lado digital, los *buffers* replicaban la señal de entrada, y la integridad de la señal se mantenía a lo largo de los trazos del sistema. Los *FETs* para los ventiladores conmutan de forma correcta. En el área de monitoreo, las señales de los sensores de corriente y voltaje de las fases eran amplificados de forma esperada, y los voltajes de salida estaban dentro del rango de los *ADCs*.

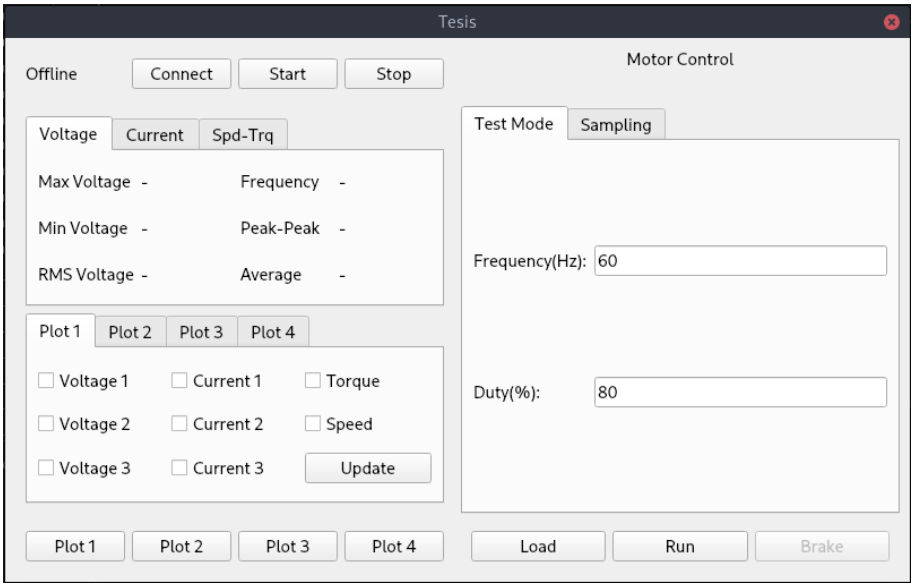


Figura 4.9: Interfaz gr´afica programada para controlar y visualizar las variables del motor.

La configuración del *FPGA* fue validada de dos formas, la primera en simulación y la segunda en la implementación física, siendo la más importante. Teniendo como punto de partida que el inversor era funcional, se corroboró que los *ADCs* fueron leídos de forma correcta, haciendo una lectura paralela en los canales. Igualmente, las fases que iban al motor semejaban una onda sinusoidal, la cual podía variar su frecuencia y su ciclo de trabajo. De igual forma, fue posible inicializar el sistema operativo que se creó con Yocto, que fue grabado en una memoria. Utilizando Linux, fue posible configurar el *FPGA* de forma dinámica. El controlador que se programó era capaz de escribir y leer de forma correcta los buses que estaban conectados al *FPGA*. Los métodos definidos para controlar el *FPGA* activaban las funciones designadas, mismas que controlaban el motor y el *ADC*. A través de peticiones, fue posible crear archivos de texto que eran empaquetados con la información de las variables del motor. En caso de introducir un bloque de control de lógica difusa, el usuario sólo deberá definir el valor deseado de velocidad y será retro-alimentado por los valores del *ADC* y las lecturas de los sensores de velocidad y torque.

Dentro de la interfaz, basada en PyQt5, fue posible implementar la creación de gráficas de forma dinámica. Definiendo un servidor y un cliente, se pudo iniciar una comunicación entre la interfaz gráfica y el sistema del *FPGA*. Los botones fueron asignados a un método que finalmente se reflejaba como una acción dentro del inversor. Dando como resultado un sistema funcional que cubre con los objetivos de este proyecto. Mismo, que tiene la capacidad de conectarse a un sistema de control difuso, que manipule las variables según se desee. En la figura 4.10 se muestra la circuitería del sistema final, que fue probado con un motor que estaba acoplado mecánicamente a un generador, con escobillas de corriente directa, con el cual se pueden realizar variaciones en la carga. En este caso, la carga constaba de una matriz de focos. La configuración mecánica se muestra en la figura 4.11.

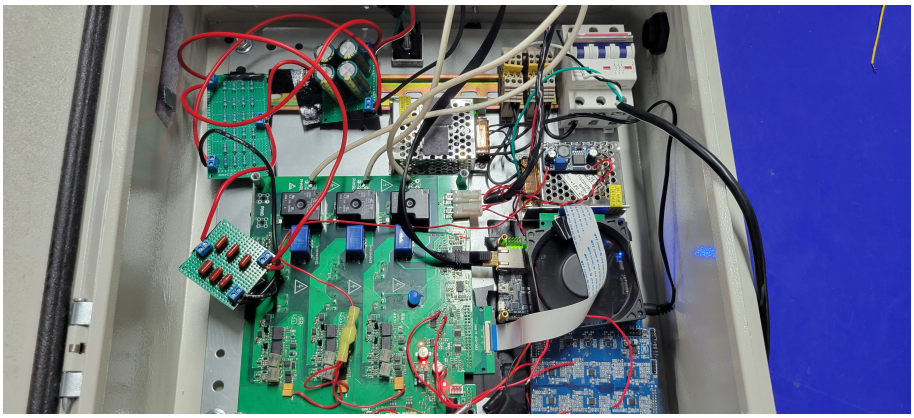


Figura 4.10: La circuiter´ıa del inversor con los componentes principales del sistema, dentro de un gabinete el´ectrico.

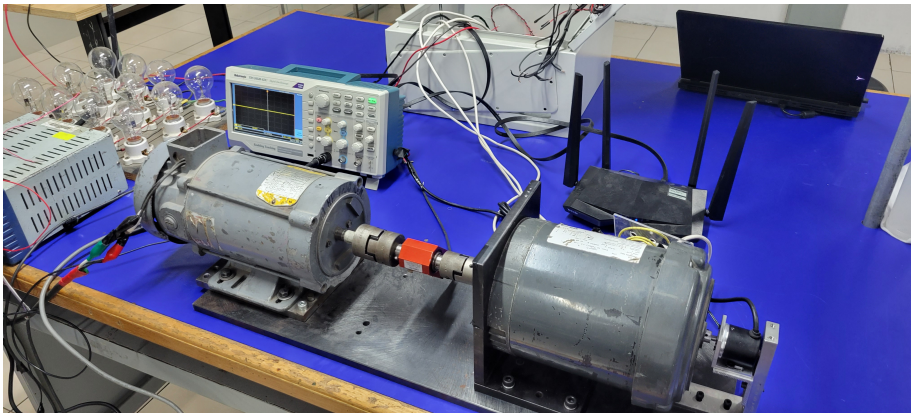


Figura 4.11: Configuraci´on mec´anica del acoplamiento de motores y sensor de torque.

4.2 Discusión

Teniendo el sistema funcional, se realizaron una serie de pruebas para validar las capacidades en conjunto. Para ello se capturaron los datos de arranque del motor a diferentes cargas. La aproximación que se utilizo fue un tanto conservadora, ya que se definió una frecuencia de conmutación para los dispositivos FET, que dependía de una onda tipo diente de sierra de $1kHz$. Dicha onda que era comparada con los valores de una función sinusoidal, con el objetivo de minimizar cualquier tipo de componente de alta frecuencia. Por otro lado, la frecuencia de muestreo se mantuvo en $5kHz$. Un factor que afectó la definición de las frecuencias, fue el desgaste que ya tenía el motor trifásico. Pues incluso poniéndole la alimentación directa con un Variac, había momentos donde no se movía. Esto creó la necesidad de disminuir la frecuencia de prueba a $30Hz$, para asegurar el movimiento del motor desde el reposo.

En la figura 4.12, se muestran las mediciones de las 3 corrientes que alimentan el circuito durante su operación en estado estable. Se puede observar que las mediciones presentan mucho ruido, mismo

que se puede originar por tanto los sensores, como durante la etapa de amplificación. Sin embargo, es posible diferenciar las señales y observar que existe un desfase de 120° entre ellas.

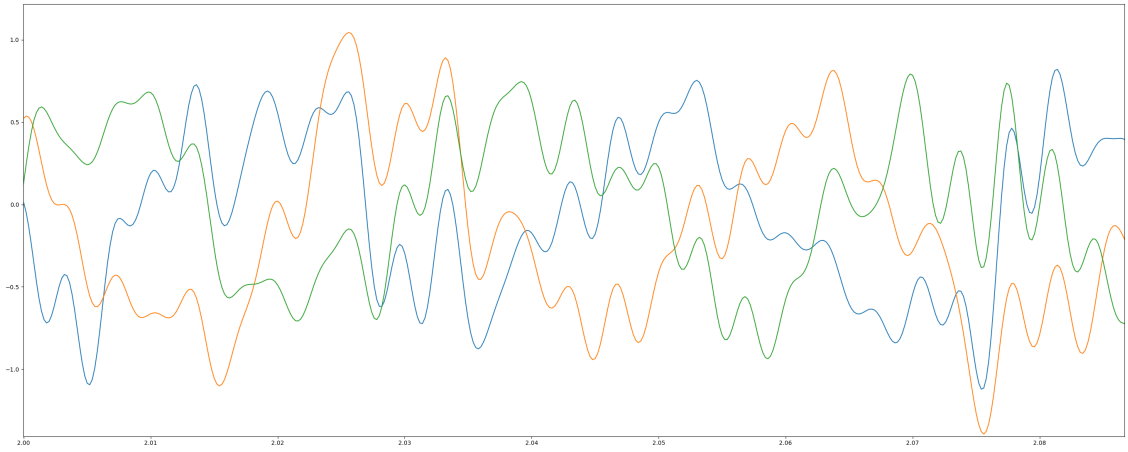


Figura 4.12: Se˜nales de corriente medidas por los sensores de efecto Hall.

Para el sistema de prueba, se utilizó una fuente de voltaje directo de $200V$ con capacidad hasta de $30A$. Para el sensor de torque, se utilizó un circuito externo que utilizaba un puente de *Wheatstone* que daba la señal con una amplitud válida para el ADC, y que estaba alimentado por una fuente de $24V$. Finalmente, la matriz de focos utilizada como carga en el generador, constaba de tres filas con cuatro bombillas de $40W$, todas conectadas en paralelo. Teniendo como resistencia total, cada fila, de 90Ω.

Para la primera prueba, se procedió a activar el sistema sin carga en el lado del generador. Es decir, que el motor sólo tendría que romper el momento teniendo como carga la fricción propia, la fricción del sensor de torque (negligible) y la fricción del generador. Como era de esperarse, el motor no tuvo problema para realizar el arranque y alcanzó su velocidad máxima en un intervalo muy corto de aproximadamente $1.5s$. La corriente de RMS de la fuente, ajustada a $200V$, fue de $0.74A$. Los resultados de las gráficas de corriente, torque y velocidad se muestran en la figuras 4.13, 4.14 y 4.15, respectivamente. Además de ser muy corto el intervalo de aceleración, se puede ver como la corriente alcanza picos que rondan poco más de $1A$ y después se estabiliza posterior a la aceleración. Lo mismo pasa con el torque, donde éste incrementa durante la aceleración y después de que se alcanza el estado estable, este se reduce. La medición del torque se presenta en voltios, ya que las condiciones del arreglo no permitieron cuantificar el valor en $N - m$.

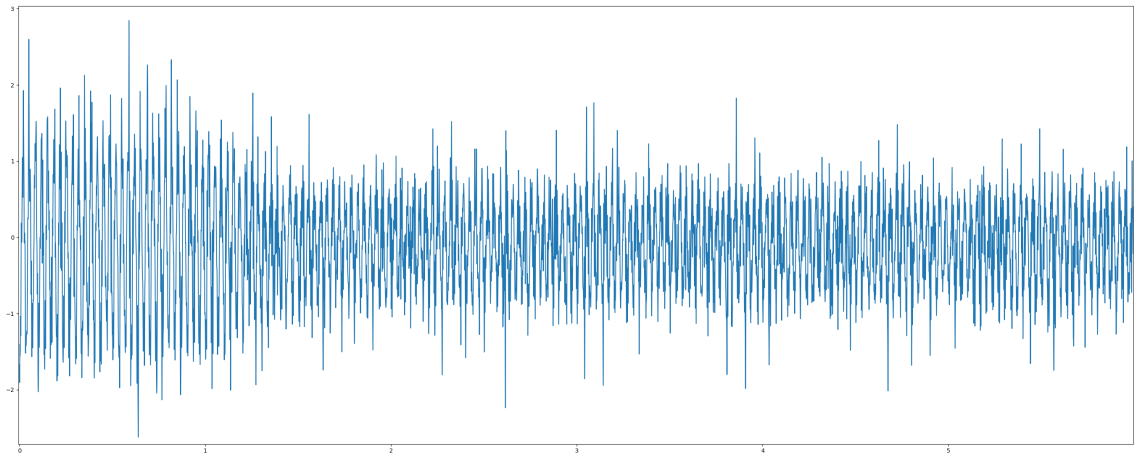


Figura 4.13: Caso 1, motor sin carga. Medici´on de corriente.

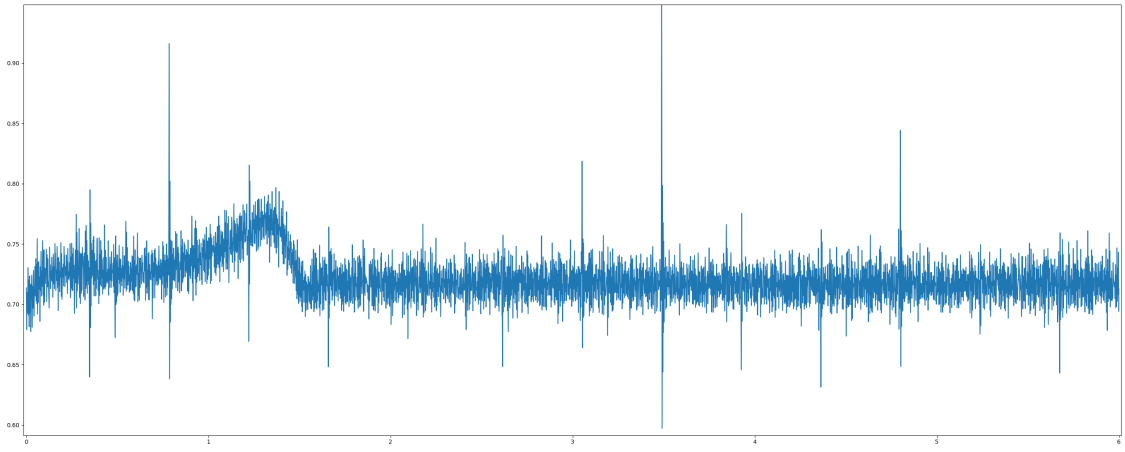


Figura 4.14: Caso 1, motor sin carga. Medici´on de torque.

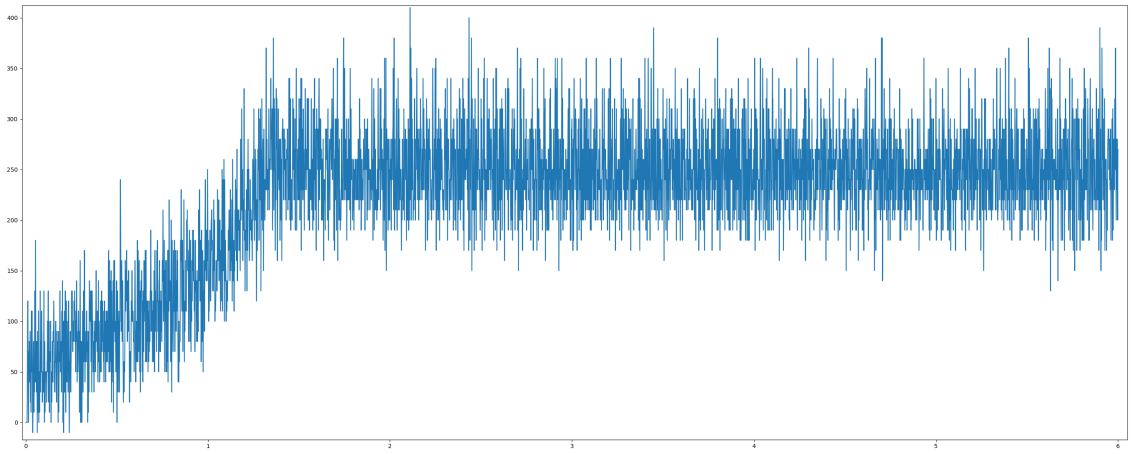


Figura 4.15: Caso 1, motor sin carga. Medici´on de velocidad.

Para la segunda prueba, se procedió a activar el sistema con una carga en el lado del generador de 30Ω, haciendo una matriz de focos en paralelo. Para esto, que el motor tendría que romper el momento teniendo como carga las fricciones, más la carga de oposición en el lado del generador mayor que el caso anterior. Aquí el motor tuvo problema para realizar el arranque y la velocidad que alcanzó fue mucho menor. Parecía por un momento que no iba a ser capaz de romper el momento. La corriente de RMS de la fuente, ajustada a $200V$, fue de $2.02A$. El brillo de los focos en este caso era bajo, apenas si brillaban los filamentos de las bombillas. Se puede observar como las corrientes en el estado estacionario, así como las del arranque fueron mayores al caso anterior. Bajo estas condiciones también el torque se vió afectado, ya que con la carga tanto en estado estacionario, como durante el arranque, los valores reportados fueron mayores.

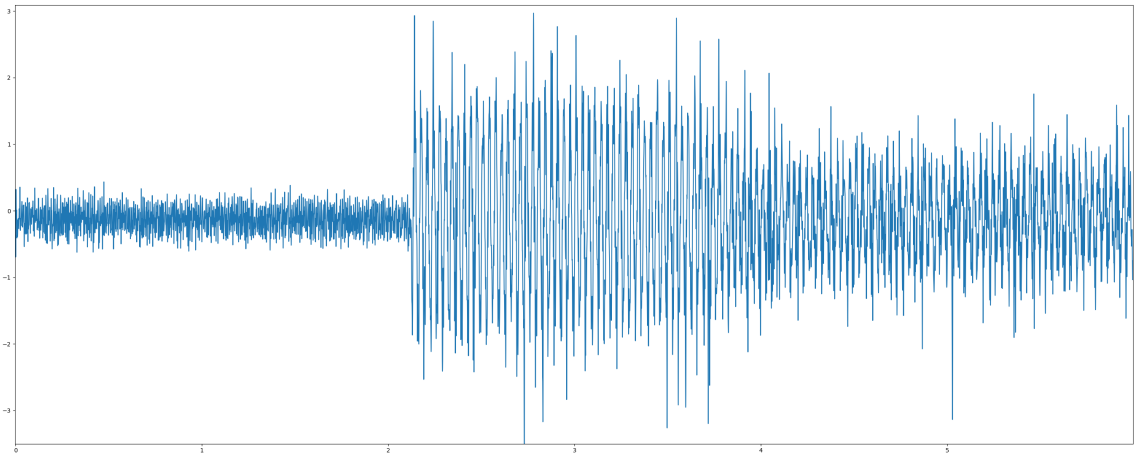


Figura 4.16: Caso 2, motor con una carga de 30Ω colocada en el generador. Medici´on de corriente.

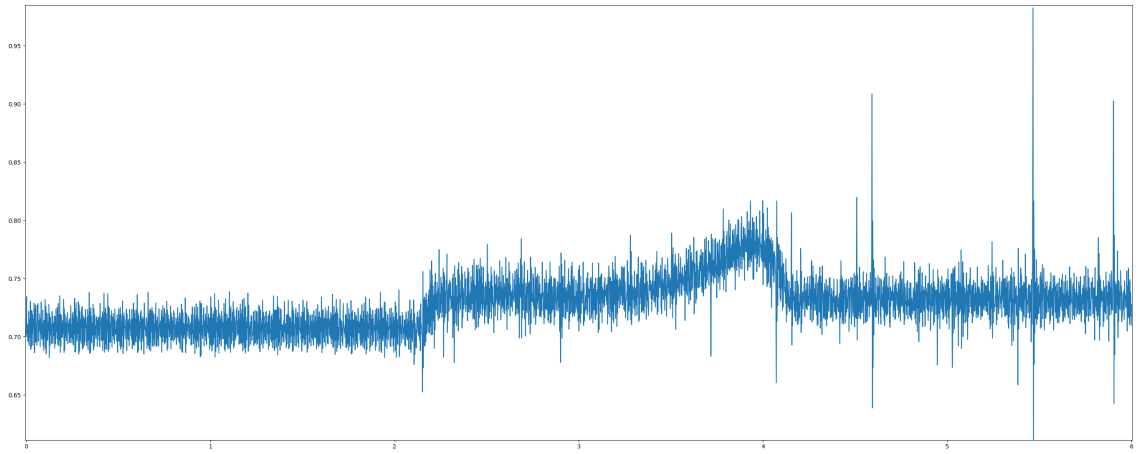


Figura 4.17: Caso 2, motor con una carga de 30Ω colocada en el generador. Medici´on de torque.

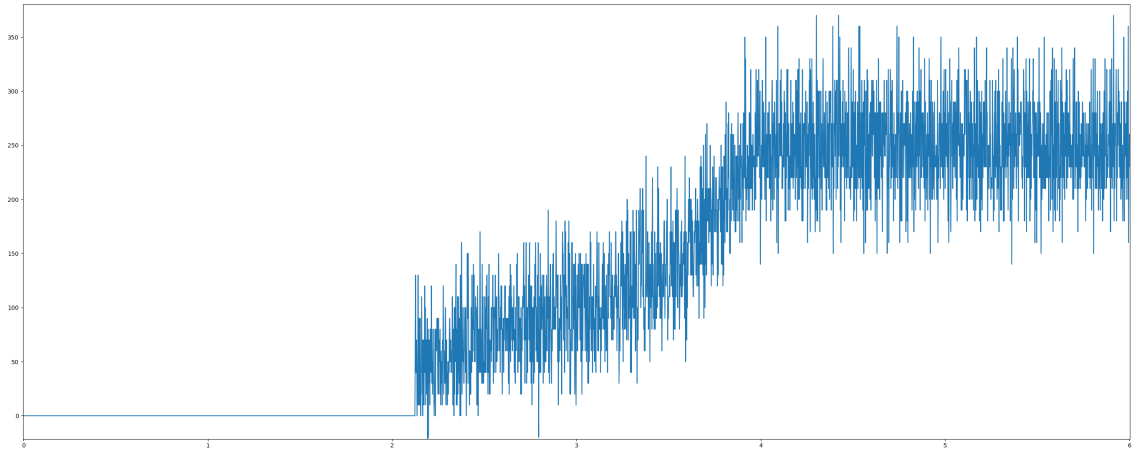


Figura 4.18: Caso 2, motor con una carga de 30Ω colocada en el generador. Medici´on de velocidad.

4.3 Impacto

La realización de éste trabajo de investigación trae consigo una serie de impactos en varios ámbitos de formas directas o indirectas. Primero, en un ámbito tecnológico, se pudo probar que la implementación de nuevas tecnologías para problemas que se han estudiado desde hace mucho tiempo, abre un área de oportunidad para tener diferentes acercamientos en el camino a la solución. En este caso puntualmente, el crear un inversor trifásico con una topología muy conocida, utilizando *FETs* con tecnología de carburo de silicio para incrementar la eficiencia del sistema y las velocidades de conmutación del sistema. Por otro lado, el utilizar un *FPGA SoC* que puede intercomunicar un *FPGA* y un procesador con un sistema operativo, maximizando la velocidad y habilitando el paralelismo con el hardware del sistema. Conservando la compatibilidad del sistema operativo, para un procesamiento posterior en cualquier computadora capaz de correr python.

Segundo, en un ámbito educativo, se pudo probar que un alumno de nivel licenciatura es capaz de comprender y diseñar sistemas complejos, de última tecnología, que puedan ser utilizados para mejorar la infraestructura de la universidad. Habilitando a nuevos estudiantes la posibilidad de experimentar y comprender esta clase de sistemas.

Tercero, en un ámbito personal y profesional, se pudo probar que es posible desarrollar habilidades, basadas en fundamentos aprendidos a lo largo de la carrera, que permiten romper con los esquemas de lo que se enseña en clase e ir más allá. Dichas habilidades permiten que el alumno pueda introducirse a un mercado profesional de manera más sencilla, con un nivel de competitividad mayor.

4.4 Trabajo futuro

Un problema que se tuvo a lo largo del desarrollo del sistema, como se puede observar en la sección de resultados de hardware, fue la constante mutación del sistema para ser mejorado. Sin duda, todas esas iteraciones consumieron bastante tiempo y surgieron con base a la mejora continua de las nuevas ideas que podían ser implementadas en el sistema. Dando como resultado final algo muy diferente a los diseños iniciales. Sin embargo, quedan algunas cosas pendientes que no fueron implementadas en el sistema que podrían mejorar su desempeño y robustez. Por ejemplo, la implementación del bloque de control difuso o cualquier tipo de controlador que permitiera variar las señales de entrada como respuesta a estímulos externos. Igualmente, la implementación de una mejor estrategia de sobre-corriente en los dispositivos, ya que durante la etapa de pruebas. Los el método de protección que consideraba las corrientes de saturación de los *FETs* era muy sensible, sobretodo a los picos de corriente al conmutar a mayores frecuencias.

Capítulo 5

Conclusión

Una vez finalizado el trabajo de investigación, se puede afirmar que es posible construir una estación de pruebas y monitoreo que es capaz de operar y visualizar variables de motores de corriente alterna trifásicos, basada en un sistema embebido de alta velocidad con una interfaz en tiempo real. Si bien, en un recorrido de más de dos años, la manera de abordar el problema fue evolucionando. Es muy notorio en la sección de los resultados de hardware, donde el proceso de detallado del circuito base muestra una mejora a través de las iteraciones. Es increíble lo mucho que se puede aprender y mejorar en tan poco tiempo, cuando se le invierten horas de desarrollo a un proyecto. En este caso, esa inversión de horas y recursos trajo como resultado la construcción de la estación didáctica que integra una interfaz de tiempo real, los elementos de control de motores, circuitos de protección y aislamiento, y múltiples sensores que recolectan variables del motor. Esta estación usa como base una tarjeta donde se integran los elementos de monitoreo y de control que son integrados a una interfaz gráfica utilizando un FPGA SoC, en este caso el Intel Cyclone V SoC. La interfaz gráfica, que sera ejecutada en la computadora del usuario mientras brinda protección y aislamiento al usuario, muestra los valores en tiempo real de las variables que se monitorean en los circuitos de senado y tiene la capacidad de suministrar alimentación a un motor de inducción trifásico.

Dentro de este desarrollo, se implementaron una serie de buenas practicas y lecciones aprendidas que son aplicadas en la industria del hardware automotriz hoy en día. Muchas veces, se tiene la tendencia de buscar abordar el problema de forma directa con los recursos que uno conoce y domina. Sin embargo, para un problema específico, como este caso, si se dedica más tiempo de planeación es posible evaluar un mayor número de opciones de como abordar el problema. Las opciones incluyen circuitos integrados que jamás se han visto ni probado, pero que tienen la suficiente documentación para generar una aproximación matemática, a través de análisis y simulaciones, que nos guiaran a tener los resultados que se buscan. De ahí, que en este proyecto se incursionó en la tecnología de carburo de silicio, y toda la circuitería que conlleva diseñar un circuito para controlar el mismo. Haciéndolo funcionar en la primera iteración. Por otro lado, el dedicar el tiempo necesario para hacer los esquemáticos y la tarjeta del circuito impreso. Siempre siguiendo las normas de diseño, cuidando las líneas que portarían mayor corriente y procurar utilizar trazos más gruesos; las señales de alta velocidad, poniendo planos de tierra entre las señales para preservar su aislamiento e integridad. Igualmente, dentro de la selección de componentes, es indispensable hacer cálculos para tener una aproximación de los valores esperados y garantizar el correcto funcionamiento de los componentes seleccionados. Por ejemplo, dentro del diseño del área de potencia, asegurar que el incremento de temperatura a la corriente máxima se encuentre dentro

del rango de temperatura de operación del componente.

Fue un gran reto entender como interconectar el sistema operativo de un procesador con un FPGA que se encontraba en el mismo circuito integrado. Posiblemente, lo que tomó un mayor tiempo de desarrollo. Dado que para poder describir el sistema, era necesario entender como funciona un sistema operativo a nivel del kernel y los protocolos de comunicación del procesador, a través de los cuales se lograría la comunicación para posteriormente, hacer funciones para programar el FPGA usando archivos binarios en Linux. Utilizando una distribución del sistema operativo personalizada, que se generó utilizando Yocto. En estas etapas, dada la falta de popularidad de esta tecnología, no hubo atajos, sólo la lectura de las hojas de aplicación y manuales de operación. En el desarrollo de tecnología, específicamente la última tecnología, es común tener que leer documentos técnicos que no han sido oficialmente lanzados y que siguen siendo escritos y corregidos.

Es muy satisfactorio el haber cumplido con la hipótesis y los objetivos de este proyecto.

Bibliografía

[1] Purkait, Biswas, Das, and Koley, *Electrical and Electronics Measurements and Instrumentation*, vol. First Edition. McGraw Hill Education, 2013.

[2] R. W. Erickson and D. Maksimovic, *Fundamentals of Power Electronics*, vol. Second Edition. Kluwer Publications, 2001.

[3] Khalil, H. K, and J. Grizzle, *Nonlinear systems*, vol. 3. Prentice hall New Jersey, 1996.

[4] H. Y. Fang Lin Luo, *Power Electronics Advanced Conversion Technologies*, vol. Second Edition. Academic Press, 2018.

[5] T. Instruments, "Power-supply design for high-speed adcs," *Application Report*, vol. Power Management, no. SLYT366A, 2010.

[6] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989.

[7] R. C. Dugan, M. F. McGranaghan, S. Santoso, and H. W. Beaty, *Electrical Power Systems Quality*, vol. First Edition. Mc Graw -Hill, 1996.

[8] A. M. Trzynadlowski, *Control of Induction Motors*, vol. First Edition. Academic Press, 2001.

[9] A. K. Maini, *Digital Electronics Principles, Devices and Applications*, vol. First Edition. Wiley, 2007.

[10] B. J. LaMeres, *Introduction to Logic Circuits & Logic Design with Verilog*, vol. Second Edition. Springer, 2019.

[11] C. M. V. Riccardo Marino, Patrizio Tomei, *Induction Motor Control Design*, vol. First Edition. Springer, 2010.

[12] Z. Gao, "Active disturbance rejection control: a paradigm shift in feedback control system design," in *2006 American control conference*, pp. 7–pp, IEEE, 2006.

[13] H. D. Tuan, P. Apkarian, T. Narikiyo, and Y. Yamamoto, "Parameterized linear matrix inequality techniques in fuzzy control system design," *IEEE Transactions on fuzzy systems*, vol. 9, no. 2, pp. 324–332, 2001.

[14] S. Herman, *Industrial Motor Control*, vol. Fifth Edition. Thomson Delmar Learning, 2005.

[15] A. Hughes, *Electric Motors and Drivers*, vol. Third Edition. Newnes, 2006.

[16] T. Wildi, *Electrical Machines, Drives and Power Systems*, vol. Sixth Edition. Prentice Hall, 2005.

[17] G. M. Rockis G., *Electrical Motor Controls*, vol. Second Edition. American Technical Publishers Inc., 2001.

[18] K. S. Tze-Fun Chan, *Applied Intelligent Control of Induction Motor Drives*, vol. Second Edition. John Wiley & Sons (Asia) Pte Ltd, 2011.

[19] S. Monk, *Programming FPGAs Getting Started with Verilog*, vol. First Edition. Mc Graw Hill, 2016.

[20] B. Ward, *How Linux Works*, vol. Second Edition. No Starch Press, 2015.

[21] R. J. Streif, *Embedded Linux Systems with the Yocto Project*, vol. First Edition. Prentice Hall, 2016.

[22] G. Kroah-Hartman, *Linux Kernel in a Nutshell*, vol. Third Edition. O'Reilly, 2006.

[23] L. T. Ya. Dorjsuren and J. Tsevegmid, "Three-axis dynamic modeling of induction motor," *Application Report*, vol. 9, no. 1998-0140, 2015.

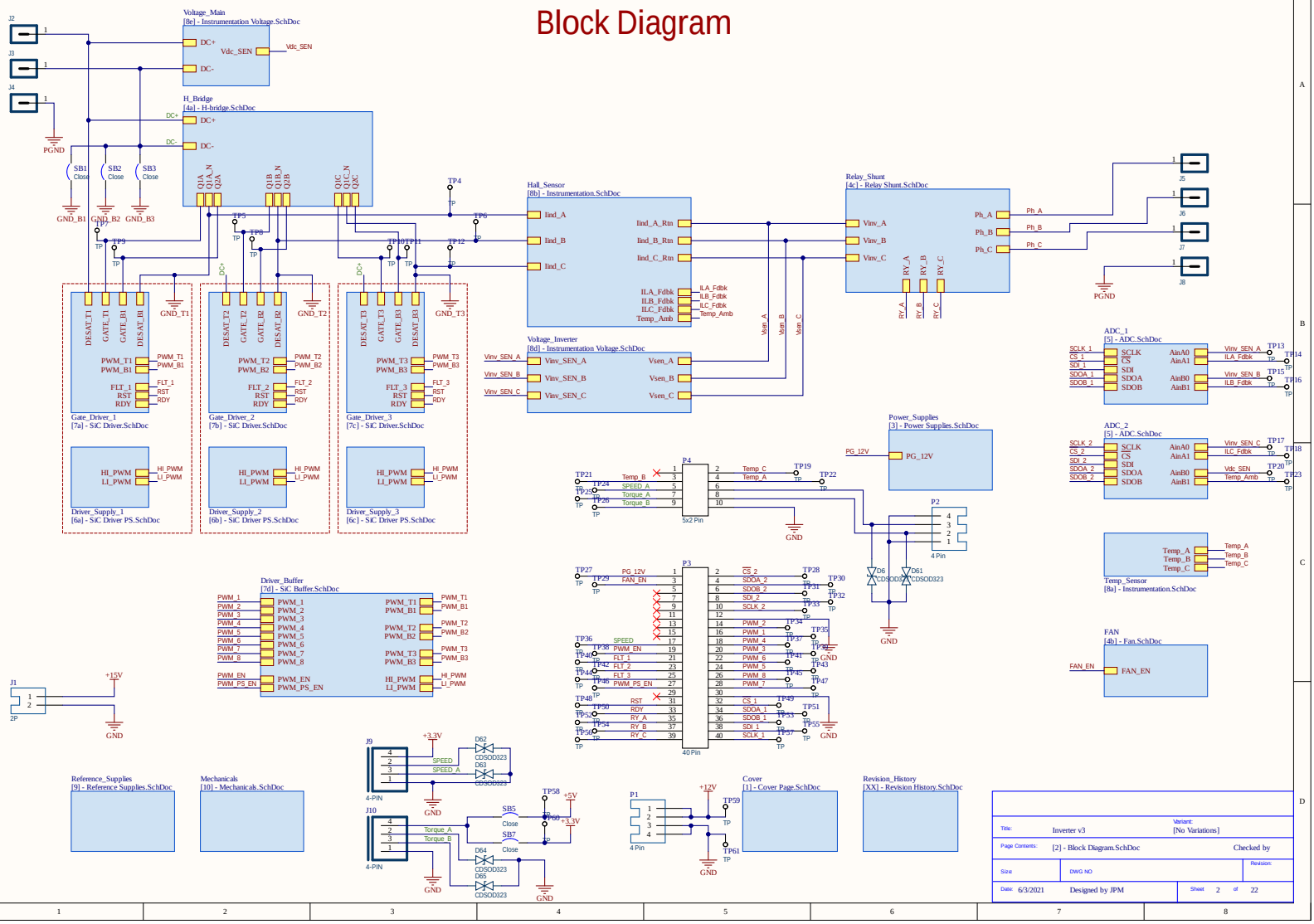
[24] X. Ramus, "Pcb layout for low distortion high-speed adc drivers," *Application Report*, vol. Power Management, no. SBAA113, 2004.

Anexo A

Apéndice

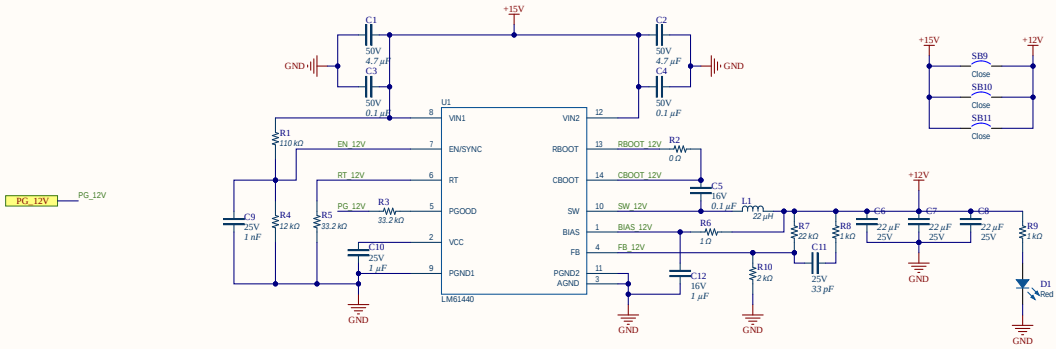
A.1 Esquemáticos

Block Diagram

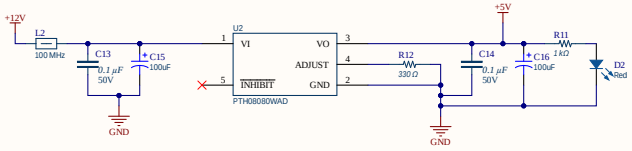


Power Supplies

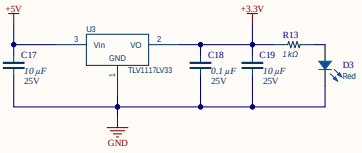
Main +12V PS



Main +5V PS

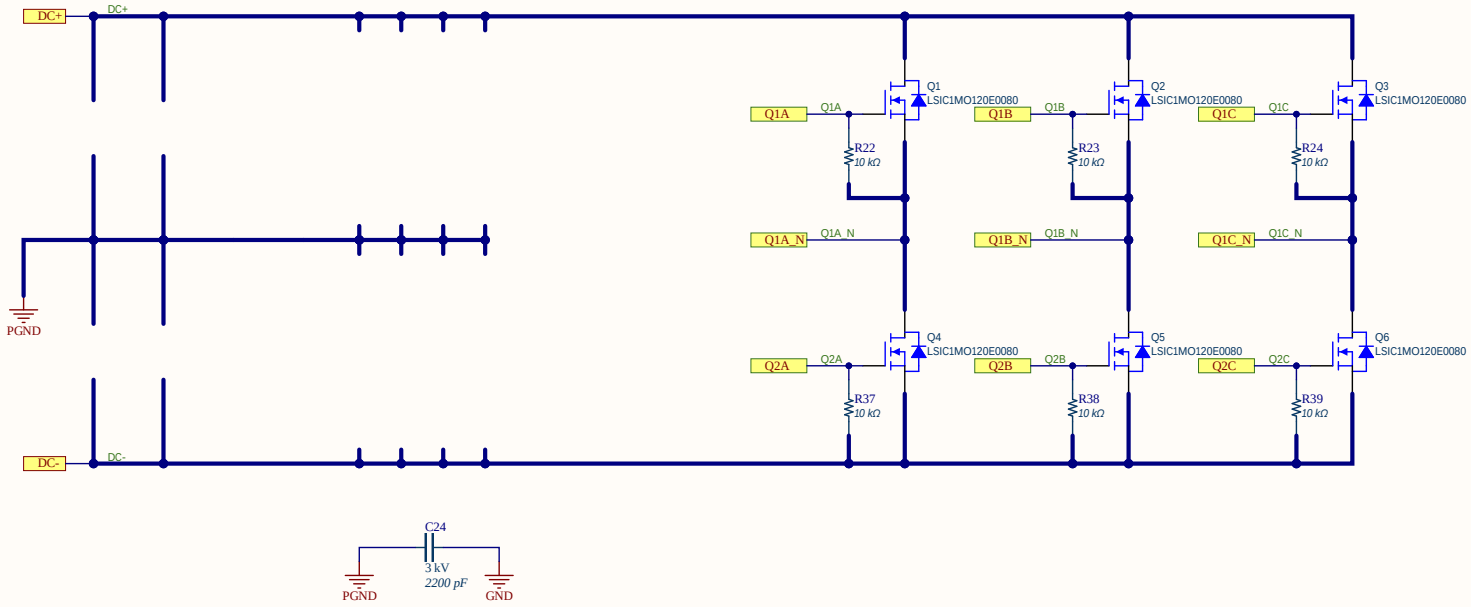


Main +3.3V PS



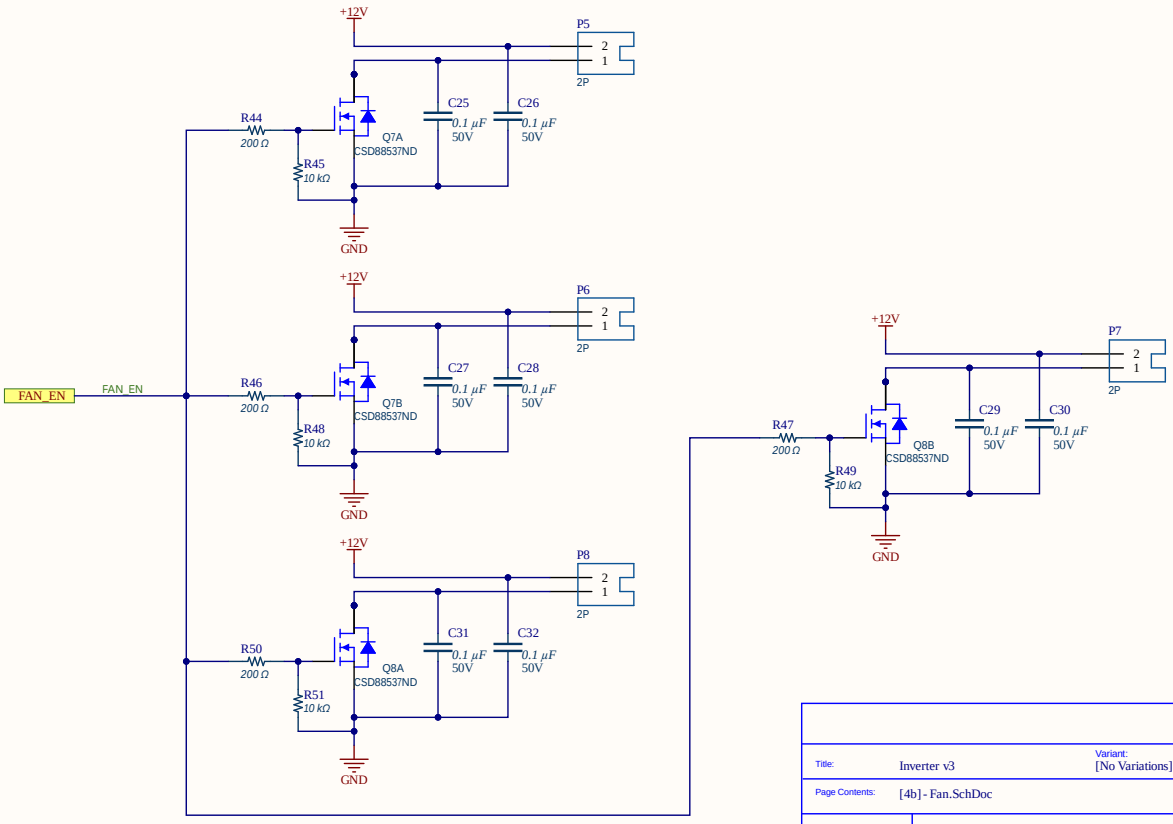
Title:	Inverter v3	Variant:	[No Variations]
Page Contents:	[3] - Power Supplies.SchDoc		Checked by
Size	DWG NO		Revision:
Date: 6/3/2021	Designed by JPM	Sheet 3	of 22

H-Bridge



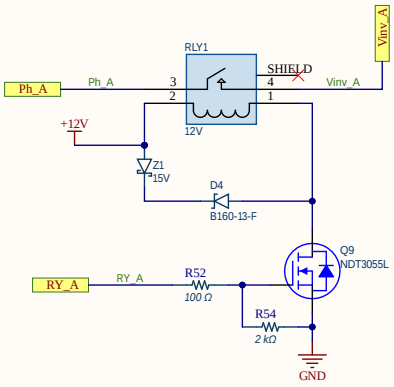
Title:	Inverter v3	Variant:	[No Variations]
Page Contents:	[4a] - H-bridge.SchDoc		Checked by
Size:	DWG NO		Revision:
Date: 6/3/2021	Designed by JPM	Sheet 4 of 22	

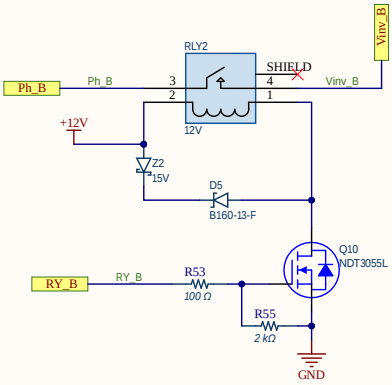
FAN

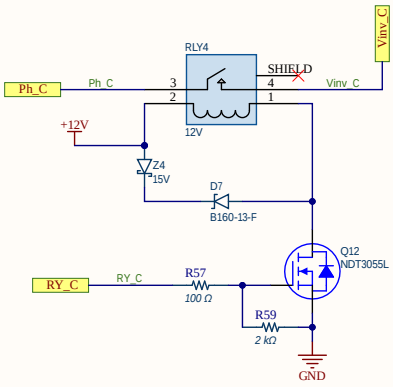


Title:	Inverter v3	Variant:	[No Variations]
Page Contents:	[4b] - Fan.SchDoc		Checked by
Size:	DWG NO		Revision:
Date: 6/3/2021	Designed by JPM	Sheet 5	of 22

Relay - Shunt

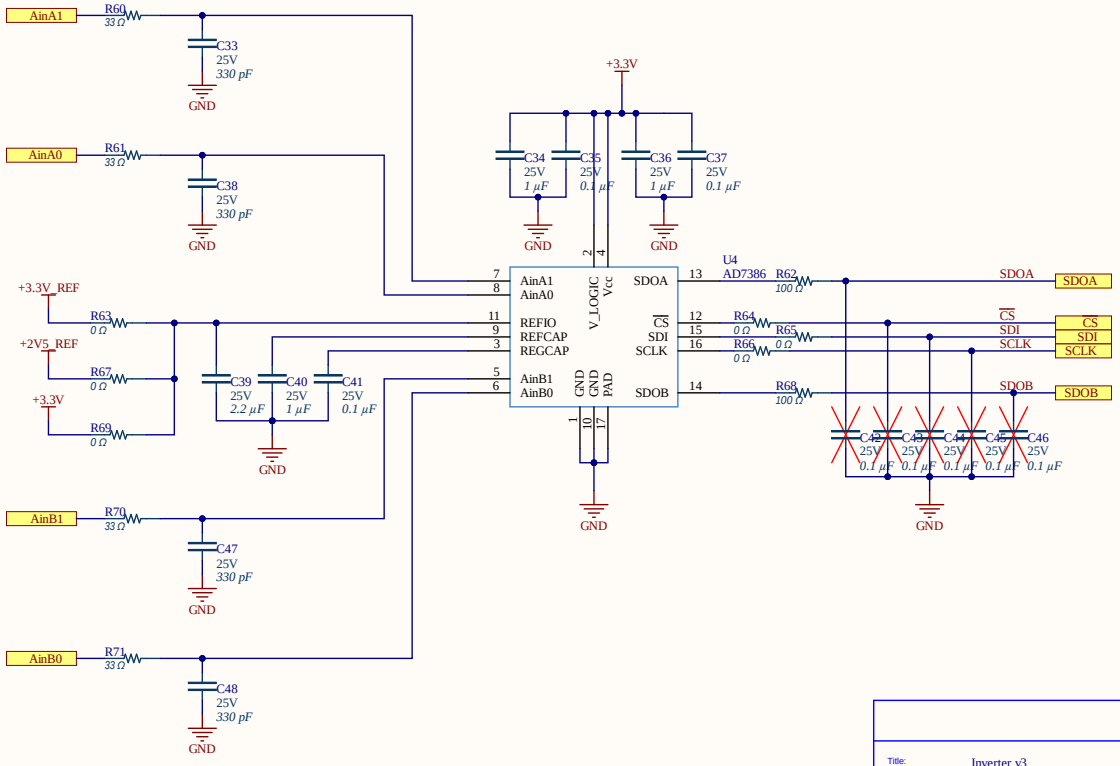






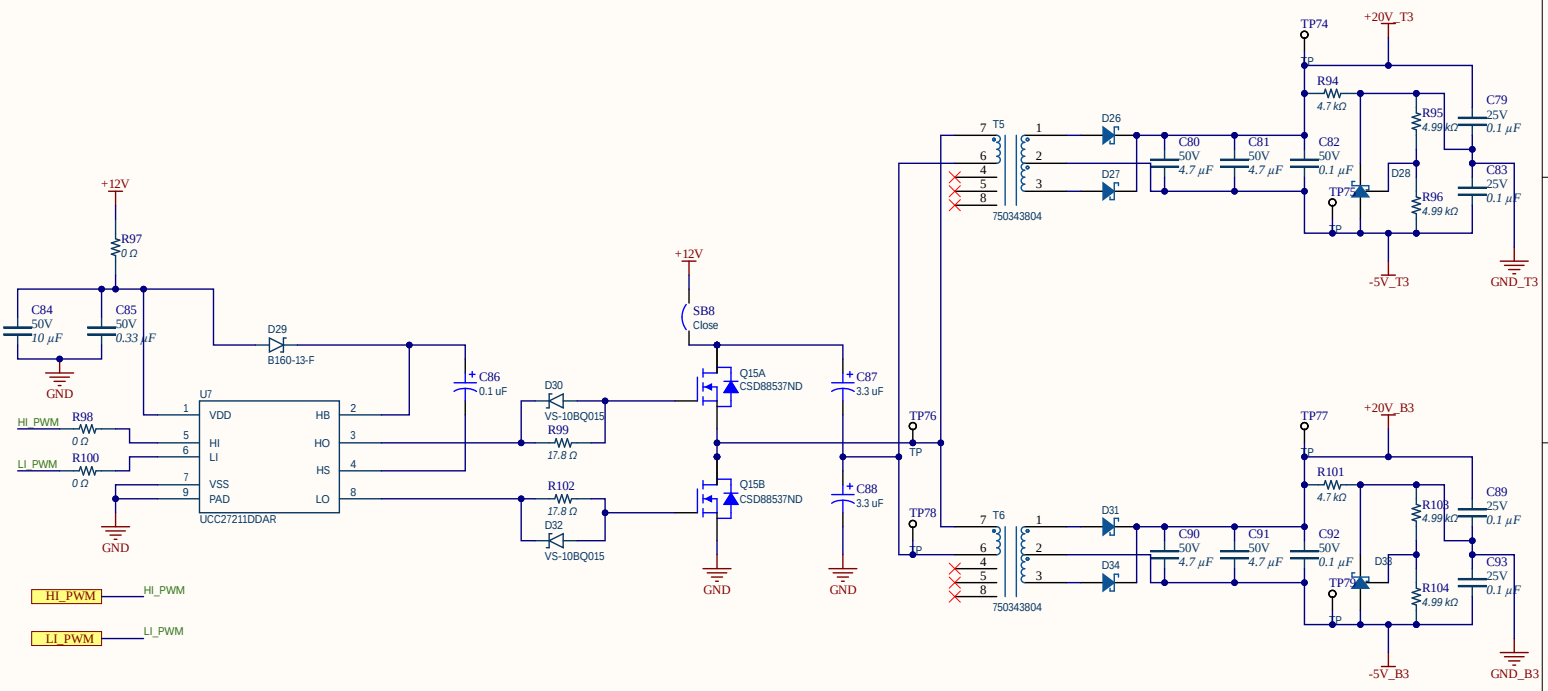
Title:	Inverter v3	Variant: [No Variations]	
Page Contents:	[4c] - Relay Shunt.SchDoc		Checked by
Size:	DWG NO		Revision:
Date: 6/3/2021	Designed by JPM	Sheet 6 of 22	

ADCs



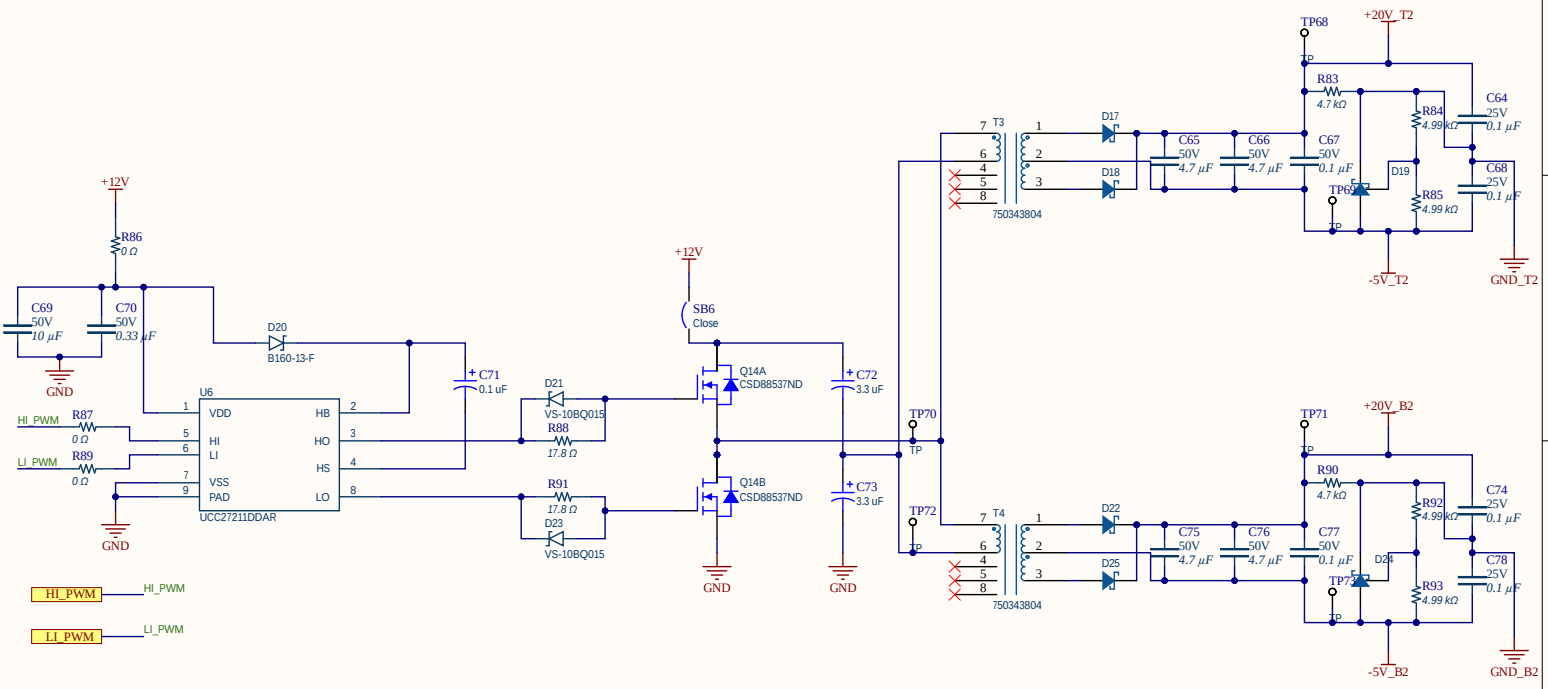
Title:	Inverter v3	Variant:	[No Variations]
Page Contents:	[5] - ADC.SchDoc		Checked by
Size:		DWG NO	Revision:
Date:	6/3/2021	Designed by JPM	Sheet 7 of 22

SiC Driver



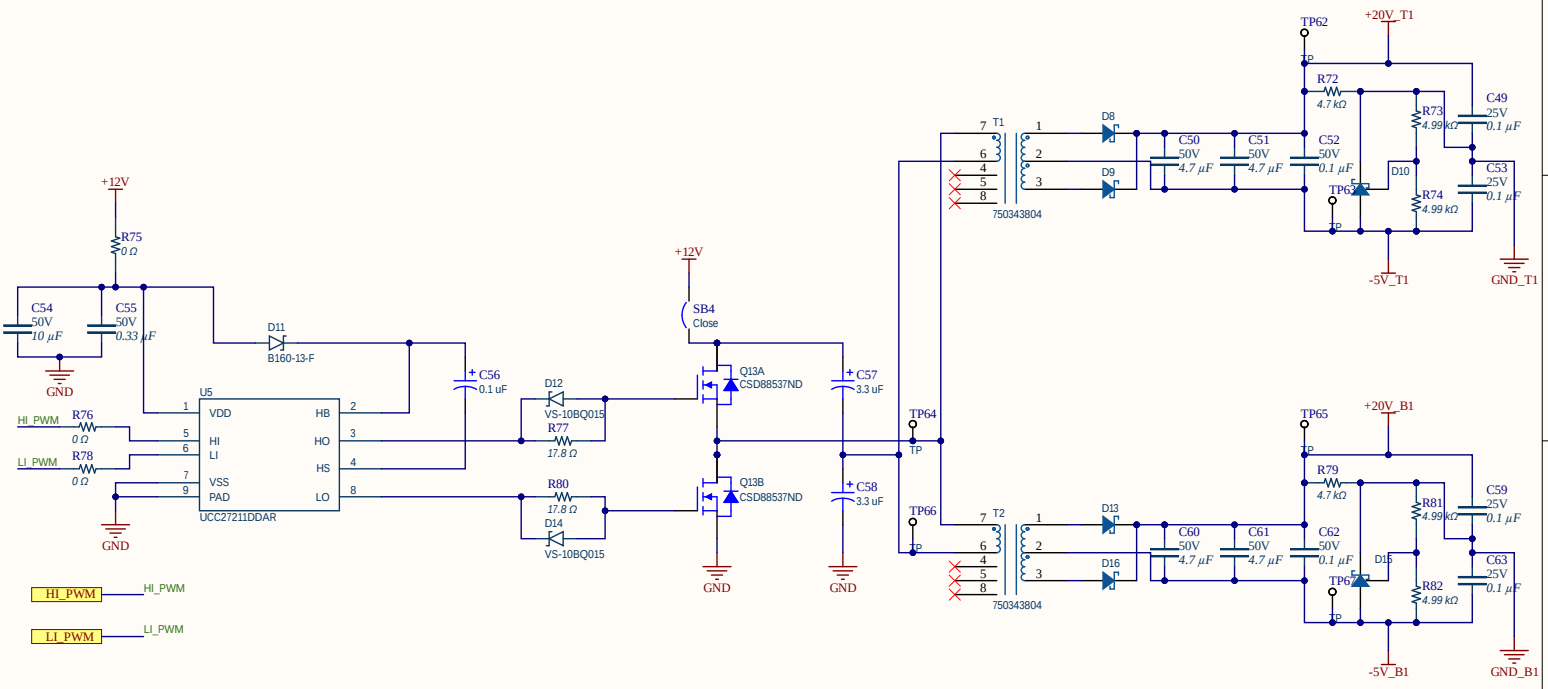
Title:	Inverter v3	Variant:	[No Variations]
Page Contents:	[6c] - SiC Driver PS.SchDoc		Checked by
Size:	DWG NO		Revision:
Date: 6/3/2021	Designed by JPM	Sheet 8 of 22	

SiC Driver



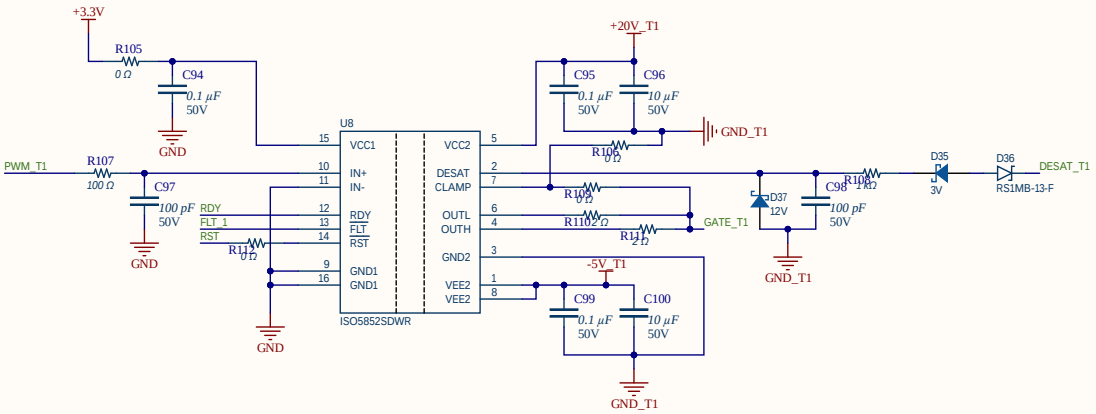
Title:	Inverter v3	Variant:	[No Variations]
Page Contents:	[6b] - SiC Driver PS.SchDoc		Checked by
Size:	DWG NO		Revision:
Date: 6/3/2021	Designed by JPM	Sheet 8 of 22	

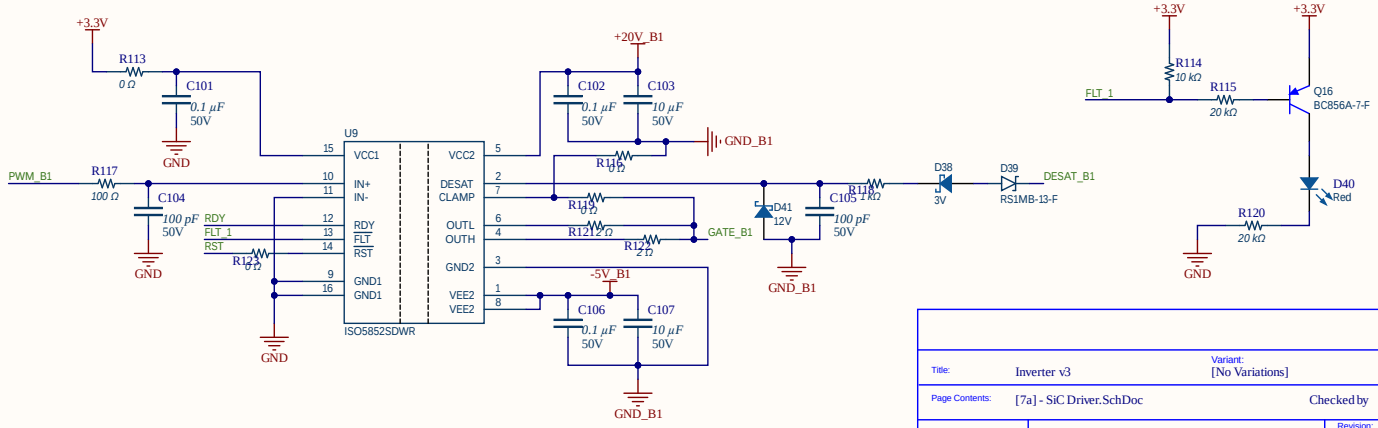
SiC Driver



Title:	Inverter v3	Variant:	[No Variations]
Page Contents:	[6a] - SiC Driver PS.SchDoc		Checked by
Size:	DWG NO		Revision:
Date: 6/3/2021	Designed by JPM	Sheet 8 of 22	

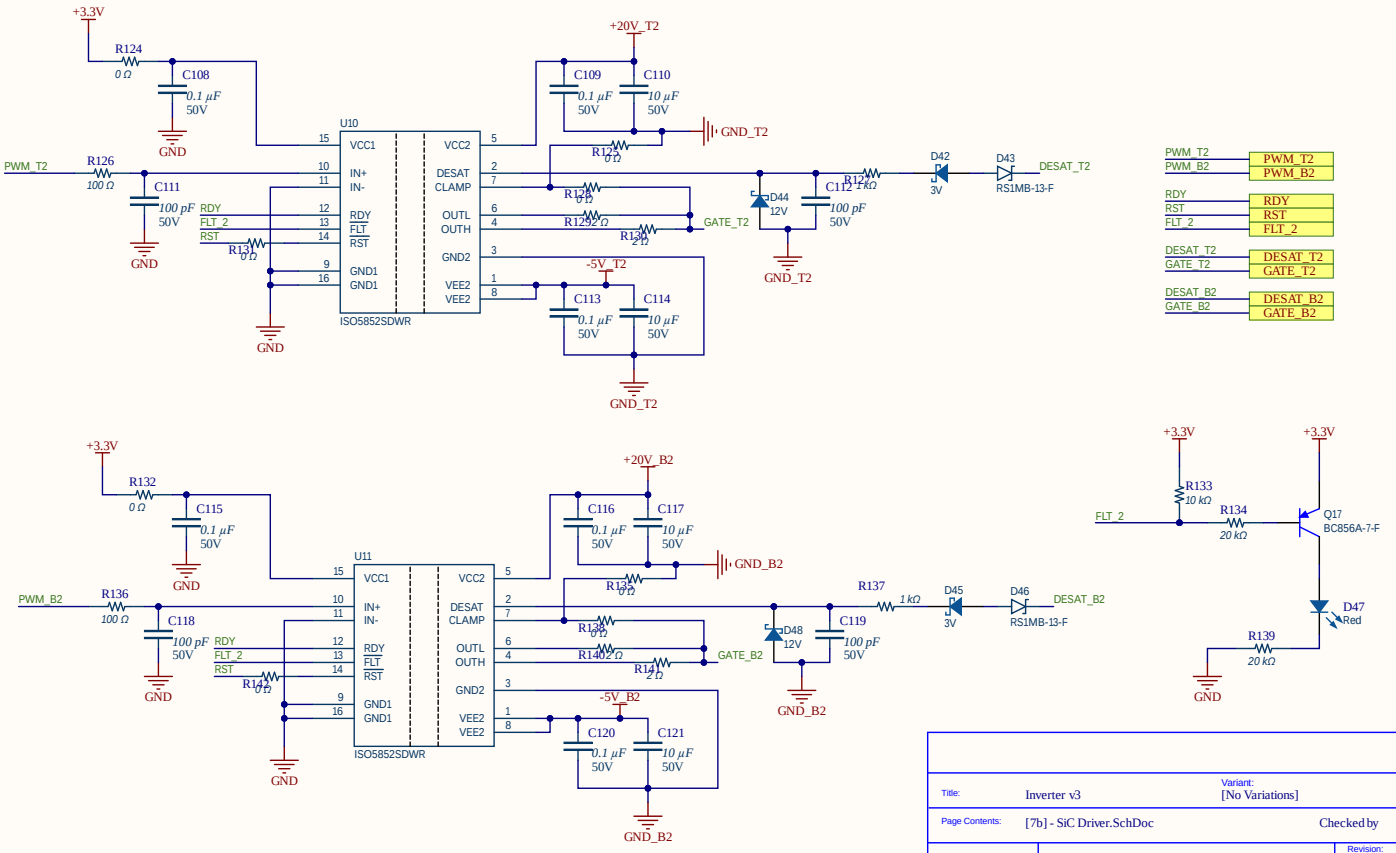
SiC Driver





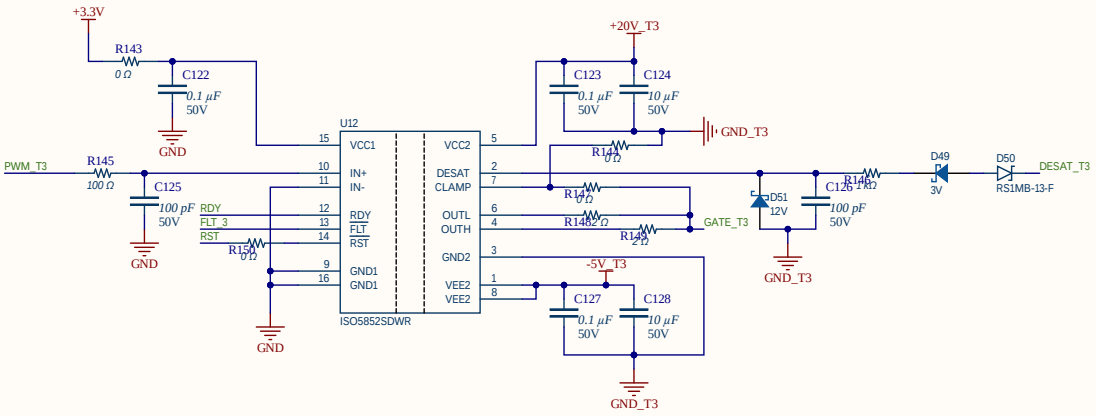
Title:	Inverter v3	Variant:	[No Variations]
Page Contents:	[7a] - SiC Driver.SchDoc		Checked by
Size:	DWG NO		Revision:
Date: 6/3/2021	Designed by JPM	Sheet 11 of 22	

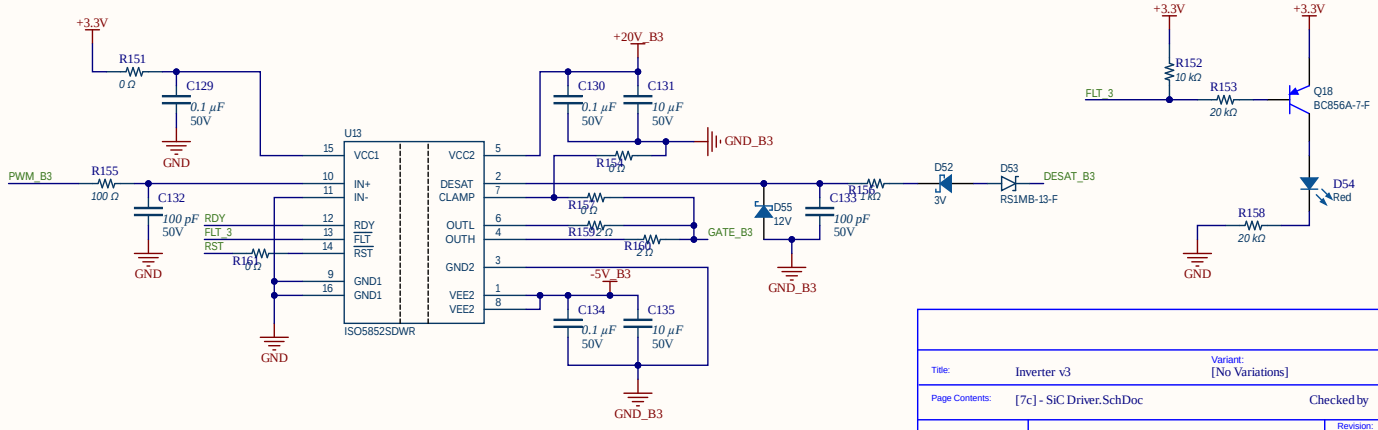
SiC Driver



Title:	Inverter v3
Variant:	[No Variations]
Page Contents:	[7b] - SiC Driver.SchDoc
Checked by	
Size:	
DWG NO	
Revision:	
Date: 6/3/2021	Designed by JPM
Sheet 12 of 22	

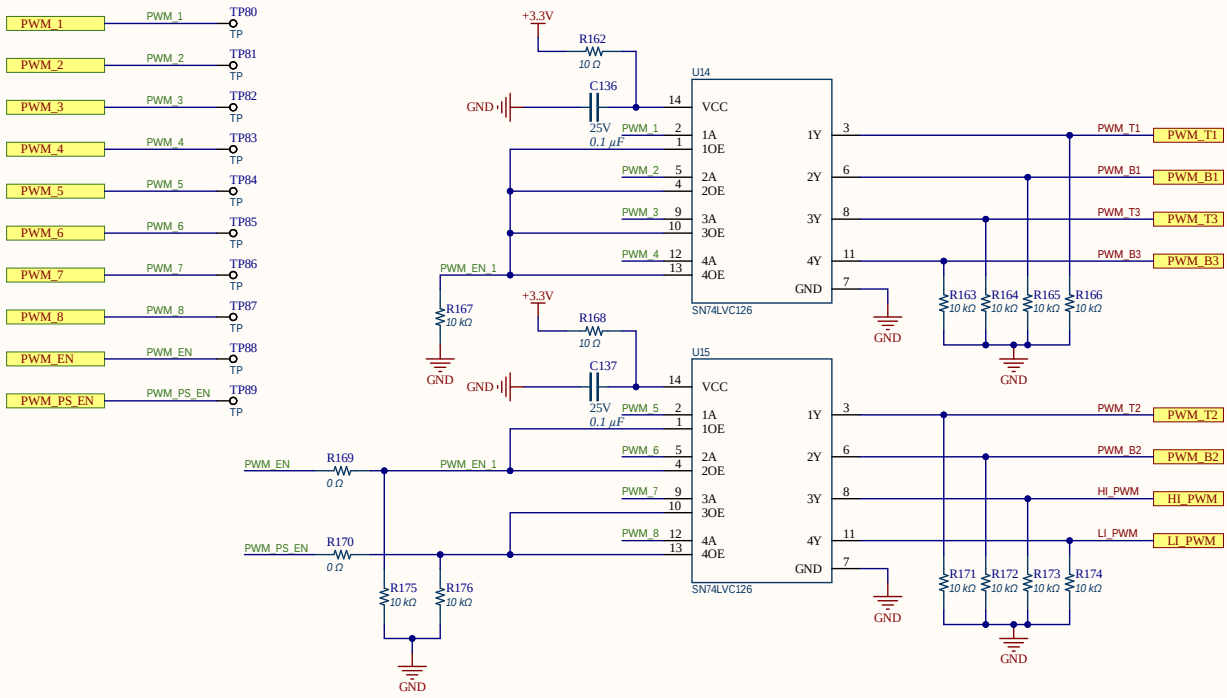
SiC Driver





Title:	Inverter v3	Variant:	[No Variations]
Page Contents:	[7c] - SiC Driver.SchDoc		Checked by
Size:	DWG NO		Revision:
Date: 6/3/2021	Designed by JPM	Sheet 13 of 22	

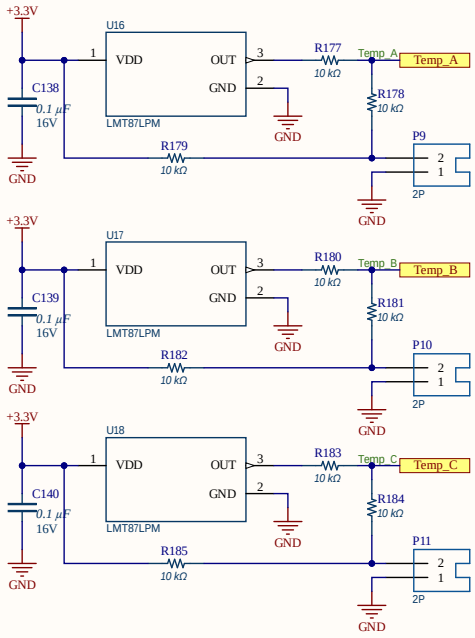
PWM Buffers



Title:	Inverter v3	Variant:	[No Variations]
Page Contents:	[7d] - SiC Buffer.SchDoc		Checked by
Size:	DWG NO		Revision:
Date: 6/3/2021	Designed by JPM	Sheet 14 of 22	

Sensors

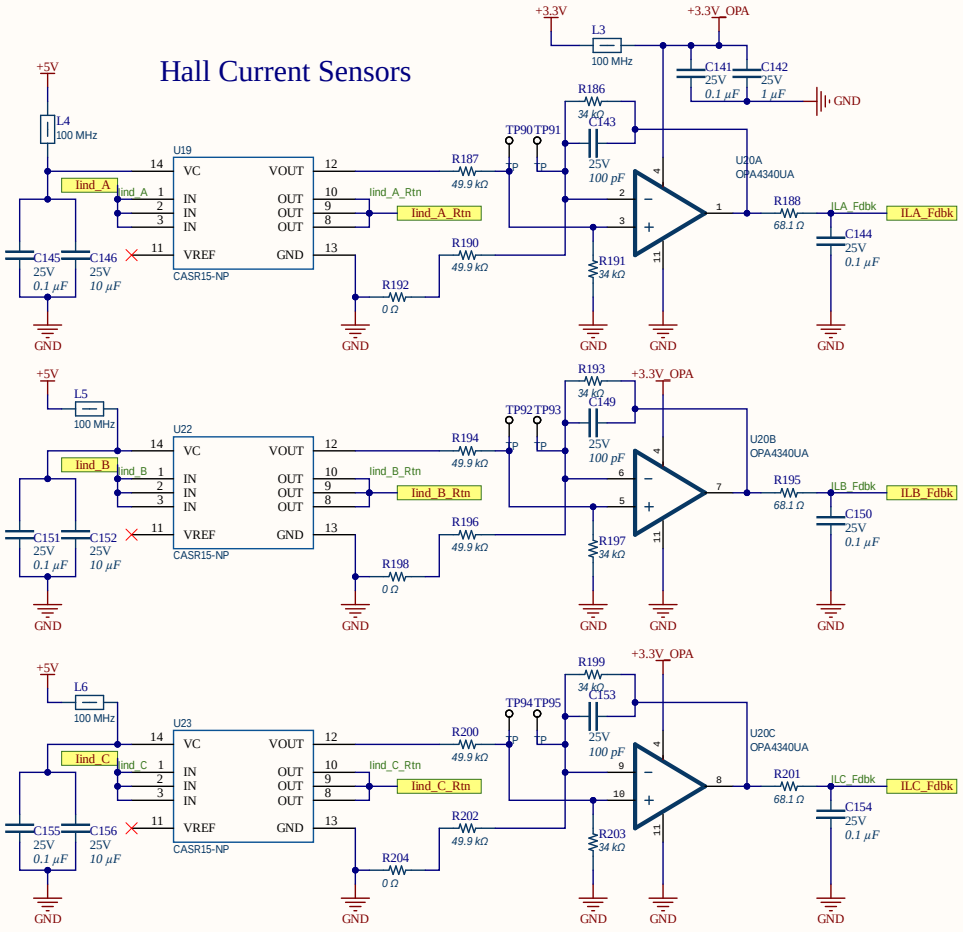
Temperature Sensors



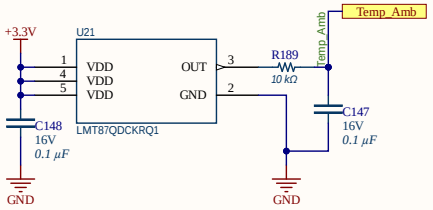
Title:	Inverter v3	Variant: [No Variations]	
Page Contents:	[8a] - Instrumentation.SchDoc		Checked by
Size:	DWG NO		Revision:
Date: 6/3/2021	Designed by JPM	Sheet 15 of 22	

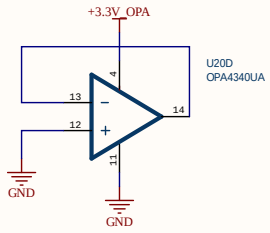
Sensors

Hall Current Sensors



Ambient Temperature

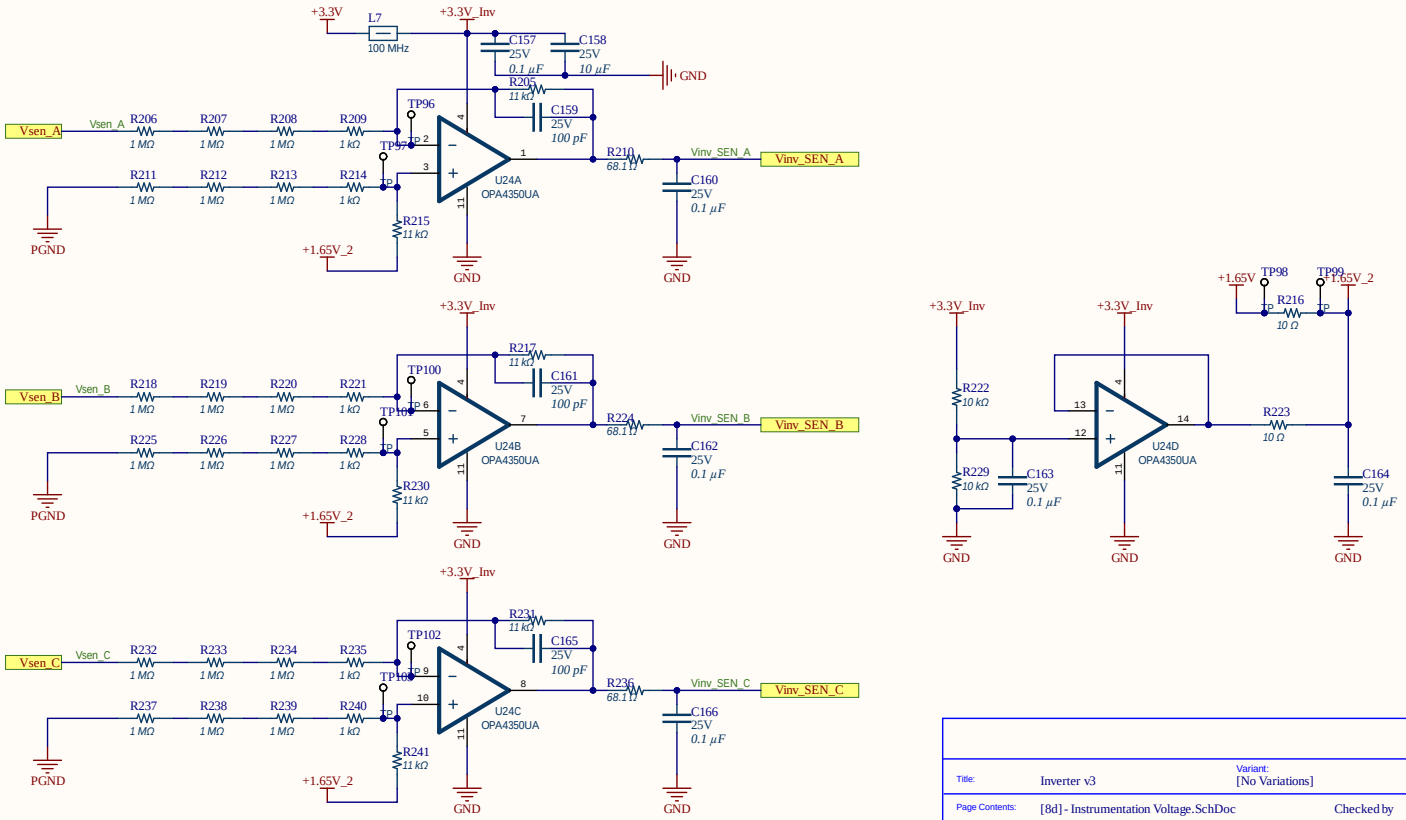




Title:	Inverter v3
Variant:	[No Variations]
Page Contents:	[8b] - Instrumentation.SchDoc
Checked by	
Size:	
DWG NO	
Revision:	
Date: 6/3/2021	Designed by JPM
Sheet 16 of 22	

Sensors

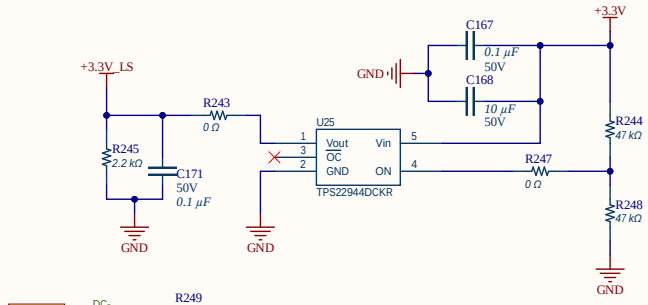
Inverter Voltage Sensors



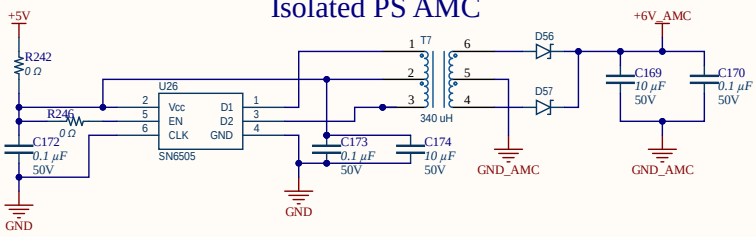
Title:	Inverter v3	Variant:	[No Variations]
Page Contents:	[8d] - Instrumentation Voltage.SchDoc		Checked by
Size:	DWG NO		Revision:
Date: 6/3/2021	Designed by JPM	Sheet 18	of 22

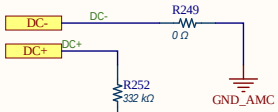
Sensors

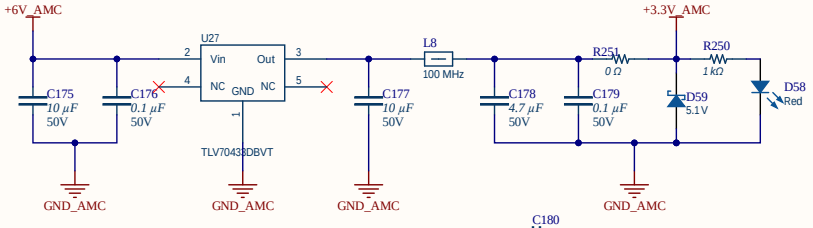
Main Voltage Sensor

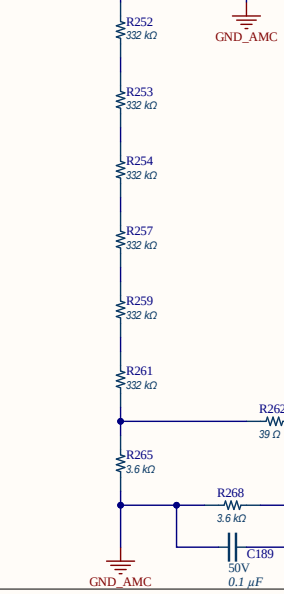


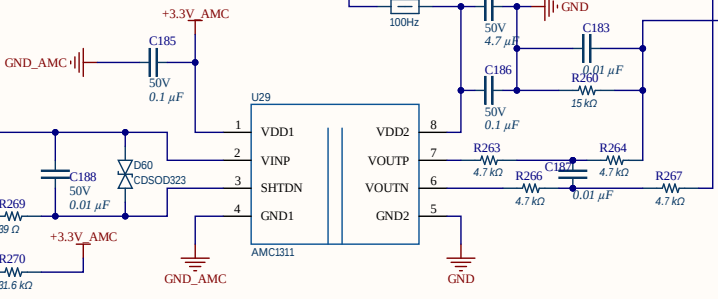
Isolated PS AMC

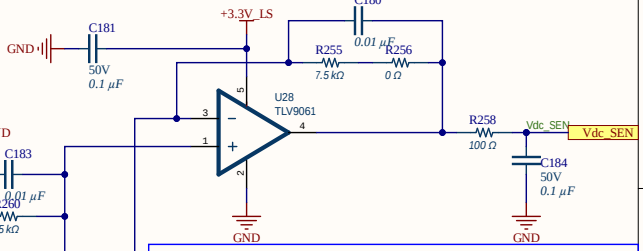






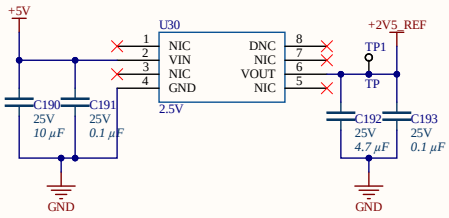


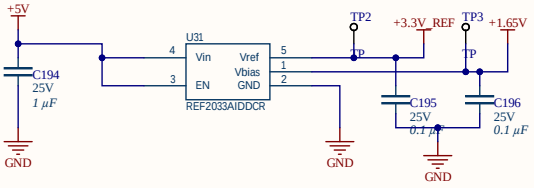




Title:	Inverter v3	Variant:	[No Variations]
Page Contents:	[8e] - Instrumentation Voltage.SchDoc		Checked by
Size:	DWG NO		Revision:
Date: 6/3/2021	Designed by JPM	Sheet 19 of 22	

Reference PS





Title:	Inverter v3	Variant: [No Variations]	
Page Contents:	[9] - Reference Supplies.SchDoc		Checked by
Size:	DWG NO		Revision:
Date: 6/3/2021	Designed by JPM	Sheet 20 of 22	

A.2 Código

A.2.1 Interfaz Gráfica

```
from pickle import TRUE
import sys
import random
import matplotlib
matplotlib.use('Qt5Agg')

from PyQt5 import QtCore, QtWidgets
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
from PyQt5.QtGui import QPixmap

from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.backends.backend_qt5agg import NavigationToolbar2QT as NavigationToolbar
from matplotlib.figure import Figure


class PlotWidget(QWidget, ):
    def __init__(self):
        super(PlotWidget, self).__init__()
        self.layout = QGridLayout()
        self.btn = QPushButton('Update', self)
        self.check_1 = QCheckBox("Voltage 1")
        self.check_2 = QCheckBox("Voltage 2")
        self.check_3 = QCheckBox("Voltage 3")
        self.check_4 = QCheckBox("Current 1")
        self.check_5 = QCheckBox("Current 2")
        self.check_6 = QCheckBox("Current 3")
        self.check_7 = QCheckBox("Torque")
        self.check_8 = QCheckBox("Speed")


        self.layout.addWidget(self.check_1,0,0)
        self.layout.addWidget(self.check_2,1,0)
        self.layout.addWidget(self.check_3,2,0)
        self.layout.addWidget(self.check_4,0,1)
        self.layout.addWidget(self.check_5,1,1)
        self.layout.addWidget(self.check_6,2,1)
        self.layout.addWidget(self.check_7,0,2)
        self.layout.addWidget(self.check_8,1,2)
        self.layout.addWidget(self.btn,2,2)
        self.setLayout(self.layout)
        #self.btn.clicked.connect(self.onClick)
```

```
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.title = 'Tesis'
        MainWindow.left = 0
        MainWindow.top = 500
        MainWindow.width = 800
        MainWindow.height = 480

        MainWindow.setObjectName(MainWindow.title)
        MainWindow.setWindowTitle(MainWindow.title)
        MainWindow.setGeometry(MainWindow.left, MainWindow.top, MainWindow.width, MainWindow.he
        MainWindow.setMinimumSize(QtCore.QSize(MainWindow.width, MainWindow.height))

        self.mainLayout = QGridLayout()
        self.statusWidget = QWidget()
        self.statusLayout = QHBoxLayout()
        self.lbStatus = QLabel("Offline")
        self.btnConnect = QPushButton("Connect")
        self.btnStart = QPushButton("Start")
        self.btnStop = QPushButton("Stop")
        self.btnPlot1 = QPushButton("Plot 1")
        self.btnPlot2 = QPushButton("Plot 2")
        self.btnPlot3 = QPushButton("Plot 3")
        self.btnPlot4 = QPushButton("Plot 4")
        self.tabwidget = QTabWidget()
        self.tabwidget.resize(300,100)
        self.voltageWidget = QWidget()
        self.voltageLayout = QGridLayout()
        self.voltageLayout.addWidget(QLabel("Max Voltage"),0,0)
        self.voltageLayout.addWidget(QLabel("Min Voltage"),1,0)
        self.voltageLayout.addWidget(QLabel("RMS Voltage"),2,0)
        self.maxVoltage = QLabel("-")
        self.minVoltage = QLabel("-")
        self.rmsVoltage = QLabel("-")
        self.voltageLayout.addWidget(self.maxVoltage,0,1)
        self.voltageLayout.addWidget(self.minVoltage,1,1)
        self.voltageLayout.addWidget(self.rmsVoltage,2,1)
        self.voltageLayout.addWidget(QLabel("Frequency"),0,2)
        self.voltageLayout.addWidget(QLabel("Peak-Peak"),1,2)
        self.voltageLayout.addWidget(QLabel("Average"),2,2)
        self.voltFrequency = QLabel("-")
        self.ptopVoltage = QLabel("-")
        self.avgVoltage = QLabel("-")
        self.voltageLayout.addWidget(self.voltFrequency,0,3)
```

```
self.voltageLayout.addWidget(self.ptopVoltage,1,3)
self.voltageLayout.addWidget(self.avgVoltage,2,3)
self.voltageWidget.setLayout(self.voltageLayout)
self.currentWidget = QWidget()
self.currentLayout = QGridLayout()
self.currentLayout.addWidget(QLabel("Max Current"),0,0)
self.currentLayout.addWidget(QLabel("Min Current"),1,0)
self.currentLayout.addWidget(QLabel("RMS Current"),2,0)
self.maxCurrent = QLabel("-")
self.minCurrent = QLabel("-")
self.rmsCurrent = QLabel("-")
self.currentLayout.addWidget(self.maxCurrent,0,1)
self.currentLayout.addWidget(self.minCurrent,1,1)
self.currentLayout.addWidget(self.rmsCurrent,2,1)
self.currentLayout.addWidget(QLabel("Ph1 Current"),0,2)
self.currentLayout.addWidget(QLabel("Ph2 Current"),1,2)
self.currentLayout.addWidget(QLabel("Ph3 Current"),2,2)
self.ph1Current = QLabel("-")
self.ph2Current = QLabel("-")
self.ph3Current = QLabel("-")
self.currentLayout.addWidget(self.ph1Current,0,3)
self.currentLayout.addWidget(self.ph2Current,1,3)
self.currentLayout.addWidget(self.ph3Current,2,3)
self.currentWidget.setLayout(self.currentLayout)
self.spdTrqWidget = QWidget()
self.spdTrqLayout = QGridLayout()
self.spdTrqLayout.addWidget(QLabel("Max Speed"),0,0)
self.spdTrqLayout.addWidget(QLabel("Min Speed"),1,0)
self.spdTrqLayout.addWidget(QLabel("Av. Speed"),2,0)
self.maxSpeed = QLabel("-")
self.minSpeed = QLabel("-")
self.avgSpeed = QLabel("-")
self.spdTrqLayout.addWidget(self.maxSpeed,0,1)
self.spdTrqLayout.addWidget(self.minSpeed,1,1)
self.spdTrqLayout.addWidget(self.avgSpeed,2,1)
self.spdTrqLayout.addWidget(QLabel("Max Torque"),0,2)
self.spdTrqLayout.addWidget(QLabel("Min Torque"),1,2)
self.spdTrqLayout.addWidget(QLabel("Av. Torque"),2,2)
self.maxTorque = QLabel("-")
self.minTorque = QLabel("-")
self.avgTorque = QLabel("-")
self.spdTrqLayout.addWidget(self.maxTorque,0,3)
self.spdTrqLayout.addWidget(self.minTorque,1,3)
self.spdTrqLayout.addWidget(self.avgTorque,2,3)
self.spdTrqWidget.setLayout(self.spdTrqLayout)
```

```
self.menu = QWidget()
self.tabwidget_2 = QTabWidget()
self.tabwidget_2.resize(300,100)
self.tabPlot1 = PlotWidget()
self.tabPlot2 = PlotWidget()
self.tabPlot3 = PlotWidget()
self.tabPlot4 = PlotWidget()
self.l = QHBoxLayout()
self.tabwidget.addTab(self.voltageWidget, "Voltage")
self.tabwidget.addTab(self.currentWidget, "Current")
self.tabwidget.addTab(self.spdTrqWidget, "Spd-Trq")
self.tabwidget_2.addTab(self.tabPlot1, "Plot 1")
self.tabwidget_2.addTab(self.tabPlot2, "Plot 2")
self.tabwidget_2.addTab(self.tabPlot3, "Plot 3")
self.tabwidget_2.addTab(self.tabPlot4, "Plot 4")

self.statusLayout.addWidget(self.lbStatus)
self.statusLayout.addWidget(self.btnConnect)
self.statusLayout.addWidget(self.btnStart)
self.statusLayout.addWidget(self.btnStop)

self.l.addWidget(self.btnPlot1)
self.l.addWidget(self.btnPlot2)
self.l.addWidget(self.btnPlot3)
self.l.addWidget(self.btnPlot4)

self.controlLabel = QLabel("Motor Control")
self.controlLabel.setAlignment(Qt.AlignHCenter)
self.ctrlTabWidget = QTabWidget()

self.motorParamWidget = QWidget()
self.motorParamLayout = QGridLayout()
self.motorParam1Label = QLabel("Frequency(Hz):")
self.motorParam1Edit = QLineEdit()
self.motorParam1Edit.setText("60")
self.motorParam2Label = QLabel("Duty(%):")
self.motorParam2Edit = QLineEdit()
self.motorParam2Edit.setText("80")

self.sampleParamWidget = QWidget()
self.sampleParamLayout = QGridLayout()
self.radial_1seg = QRadioButton()
self.radial_6seg = QRadioButton()
self.radial_6seg.setChecked(True)
```

```
self.radial_1seg_Label = QLabel("Sample every 1 sec")
self.radial_6seg_Label = QLabel("Sample every 6 sec")
self.SamplingLabel = QLabel("Sampling:")

self.motorParamLayout.addWidget(self.motorParam1Label,0,0)
self.motorParamLayout.addWidget(self.motorParam1Edit,0,1)
self.motorParamLayout.addWidget(self.motorParam2Label,1,0)
self.motorParamLayout.addWidget(self.motorParam2Edit,1,1)
self.motorParamWidget.setLayout(self.motorParamLayout)

self.sampleParamLayout.addWidget(self.SamplingLabel,0,0)
self.sampleParamLayout.addWidget(self.radial_1seg_Label,1,0)
self.sampleParamLayout.addWidget(self.radial_1seg,1,1)
self.sampleParamLayout.addWidget(self.radial_6seg_Label,2,0)
self.sampleParamLayout.addWidget(self.radial_6seg,2,1)
self.sampleParamWidget.setLayout(self.sampleParamLayout)

self.ctrlTabWidget.addTab(self.motorParamWidget, "Test Mode")
self.ctrlTabWidget.addTab(self.sampleParamWidget, "Sampling")

self.controlBtnWidget = QWidget()
self.controlBtnLayout = QHBoxLayout()
self.loadCtrlBtn = QPushButton("Load")
self.runCtrlBtn = QPushButton("Run")
self.brakeCtrlBtn = QPushButton("Brake")
self.brakeCtrlBtn.setEnabled(False)
self.controlBtnWidget.setLayout(self.controlBtnLayout)
self.controlBtnLayout.addWidget(self.loadCtrlBtn)
self.controlBtnLayout.addWidget(self.runCtrlBtn)
self.controlBtnLayout.addWidget(self.brakeCtrlBtn)

self.fixWidget = QWidget()
self.fixstatusLayout = QVBoxLayout()
self.fixstatusLayout.addWidget(self.tabwidget)
self.fixstatusLayout.addWidget(self.tabwidget_2)
self.fixWidget.setLayout(self.fixstatusLayout)

self.statusWidget.setLayout(self.statusLayout)
self.menu.setLayout(self.l)
self.mainLayout.addWidget(self.statusWidget, 0, 0)
self.mainLayout.addWidget(self.fixWidget, 1, 0)
self.mainLayout.addWidget(self.menu, 2, 0)
self.mainLayout.addWidget(self.controlLabel, 0, 1)
self.mainLayout.addWidget(self.ctrlTabWidget, 1, 1)
self.mainLayout.addWidget(self.controlBtnWidget, 2, 1)
```

```
        self.widget = QWidget()
        self.widget.setLayout(self.mainLayout)
        MainWindow.setCentralWidget(self.widget)

        QtCore.QMetaObject.connectSlotsByName(MainWindow)

if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

A.2.2 Top Level del HDL

```
‘define USE_HPS
‘define USE_ADUINO
‘define USE_ADC
// ‘define USE_HDMI
 ‘define USE_GPIO0
 ‘define USE_GPIO1

module DE10NANOyocto(

///////////////////////////////////////////////// ADC ////////////////
‘ifdef USE_ADC
output            ADC_CONVST,
output            ADC_SCK,
output            ADC_SDI,
input             ADC_SDO,
‘endif

///////////////////////////////////////////// ARDUINO //////////////
‘ifdef USE_ADUINO
inout      [15:0] ARDUINO_IO,
inout             ARDUINO_RESET_N,
‘endif

///////////////////////////////////////////// HDMI ////////////////
‘ifdef USE_HDMI
inout             HDMI_I2C_SCL,
```

```
inout             HDMI_I2C_SDA,
inout             HDMI_I2S,
inout             HDMI_LRCLK,
inout             HDMI_MCLK,
inout             HDMI_SCLK,
output            HDMI_TX_CLK,
output            HDMI_TX_DE,
output     [23:0] HDMI_TX_D,
output            HDMI_TX_HS,
input             HDMI_TX_INT,
output            HDMI_TX_VS,
`endif

//////////////////////////////////////////// HPS /////////////////
`ifdef USE_HPS
//inout              HPS_CONV_USB_N,
output     [14:0] HPS_DDR3_ADDR,
output      [2:0] HPS_DDR3_BA,
output            HPS_DDR3_CAS_N,
output            HPS_DDR3_CKE,
output            HPS_DDR3_CK_N,
output            HPS_DDR3_CK_P,
output            HPS_DDR3_CS_N,
output      [3:0] HPS_DDR3_DM,
inout      [31:0] HPS_DDR3_DQ,
inout       [3:0] HPS_DDR3_DQS_N,
inout       [3:0] HPS_DDR3_DQS_P,
output            HPS_DDR3_ODT,
output            HPS_DDR3_RAS_N,
output            HPS_DDR3_RESET_N,
input             HPS_DDR3_RZQ,
output            HPS_DDR3_WE_N,

output            HPS_ENET_GTX_CLK,
inout             HPS_ENET_INT_N,
output            HPS_ENET_MDC,
inout             HPS_ENET_MDIO,
input             HPS_ENET_RX_CLK,
input       [3:0] HPS_ENET_RX_DATA,
input             HPS_ENET_RX_DV,
output      [3:0] HPS_ENET_TX_DATA,
output            HPS_ENET_TX_EN,

inout             HPS_I2C1_SCLK,
inout             HPS_I2C1_SDAT,
```

```
inout              HPS_KEY,
inout              HPS_LED,

//inout              HPS_LTC_GPIO,
output             HPS_SD_CLK,
inout              HPS_SD_CMD,
inout       [3:0]  HPS_SD_DATA,

// output             HPS_SPIM_CLK,
// input              HPS_SPIM_MISO,
// output             HPS_SPIM_MOSI,
// inout              HPS_SPIM_SS,

input              HPS_UART_RX,
output             HPS_UART_TX,
input              HPS_USB_CLKOUT,
inout       [7:0]  HPS_USB_DATA,
input              HPS_USB_DIR,
input              HPS_USB_NXT,
output             HPS_USB_STP,
`endif

///////////////////////////////////////////// GPIO 0 ////////////////
`ifdef USE_GPIO0
inout      [35:0] GPIOGPIO,
`endif

///////////////////////////////////////////// GPIO 1 ////////////////
`ifdef USE_GPIO1
inout      [35:0] GPI1GPIO,
`endif

///////////////////////////////////////////// KEY /////////////////
input       [1:0] KEY,

///////////////////////////////////////////// LED /////////////////
output      [7:0] LED,

///////////////////////////////////////////// SW  /////////////////
input       [3:0] SW,

///////////////////////////////////////////// CLOCK /////////////////
input              FPGA_CLK1_50,
input              FPGA_CLK2_50,
```

```
input             FPGA_CLK3_50

);

//=======================================================================
//  REG/WIRE declarations
//=======================================================================

//// IO Buffer Temp I2c 1 & 3
wire scl1_o_e, sda1_o_e, scl1_o, sda1_o,
     scl3_o_e, sda3_o_e, scl3_o, sda3_o;

//// IO Buffer Temp SPI 0
wire spi0_clk, spi0_mosi, spi0_miso,spi0_ss_0_n;

//// IO Buffer Temp UART 1
wire uart1_rx,uart1_tx;

//// IO Buffer Temp CAN 0
wire can0_rx, can0_tx;

base_hps u0 (

///////////////////////////////////////////// CLOCKS /////////////////
 .clk_clk                          ( FPGA_CLK1_50 ),

////////////////////////////////////  Onboard DDR3 1GB Memmory  ///////
      .hps_0_ddr_mem_a                  ( HPS_DDR3_ADDR),
      .hps_0_ddr_mem_ba                 ( HPS_DDR3_BA),
      .hps_0_ddr_mem_ck                 ( HPS_DDR3_CK_P),
      .hps_0_ddr_mem_ck_n               ( HPS_DDR3_CK_N),
      .hps_0_ddr_mem_cke                ( HPS_DDR3_CKE),
      .hps_0_ddr_mem_cs_n               ( HPS_DDR3_CS_N),
      .hps_0_ddr_mem_ras_n              ( HPS_DDR3_RAS_N),
      .hps_0_ddr_mem_cas_n              ( HPS_DDR3_CAS_N),
      .hps_0_ddr_mem_we_n               ( HPS_DDR3_WE_N),
      .hps_0_ddr_mem_reset_n            ( HPS_DDR3_RESET_N),
      .hps_0_ddr_mem_dq                 ( HPS_DDR3_DQ),
      .hps_0_ddr_mem_dqs                ( HPS_DDR3_DQS_P),
      .hps_0_ddr_mem_dqs_n              ( HPS_DDR3_DQS_N),
      .hps_0_ddr_mem_odt                ( HPS_DDR3_ODT),
      .hps_0_ddr_mem_dm                 ( HPS_DDR3_DM),
      .hps_0_ddr_oct_rzqin              ( HPS_DDR3_RZQ),

 ////////////////////////////////////// HPS Ethernet 1  //////////
```

```
      .hps_0_io_hps_io_emac1_inst_TX_CLK ( HPS_ENET_GTX_CLK),
      .hps_0_io_hps_io_emac1_inst_TXD0   ( HPS_ENET_TX_DATA[0] ),
      .hps_0_io_hps_io_emac1_inst_TXD1   ( HPS_ENET_TX_DATA[1] ),
      .hps_0_io_hps_io_emac1_inst_TXD2   ( HPS_ENET_TX_DATA[2] ),
      .hps_0_io_hps_io_emac1_inst_TXD3   ( HPS_ENET_TX_DATA[3] ),
      .hps_0_io_hps_io_emac1_inst_RXD0   ( HPS_ENET_RX_DATA[0] ),
      .hps_0_io_hps_io_emac1_inst_MDIO   ( HPS_ENET_MDIO ),
      .hps_0_io_hps_io_emac1_inst_MDC    ( HPS_ENET_MDC  ),
      .hps_0_io_hps_io_emac1_inst_RX_CTL ( HPS_ENET_RX_DV),
      .hps_0_io_hps_io_emac1_inst_TX_CTL ( HPS_ENET_TX_EN),
      .hps_0_io_hps_io_emac1_inst_RX_CLK ( HPS_ENET_RX_CLK),
      .hps_0_io_hps_io_emac1_inst_RXD1   ( HPS_ENET_RX_DATA[1] ),
      .hps_0_io_hps_io_emac1_inst_RXD2   ( HPS_ENET_RX_DATA[2] ),
      .hps_0_io_hps_io_emac1_inst_RXD3   ( HPS_ENET_RX_DATA[3] ),

////////////////////////////////////// SD Card Boot drive  ///////////
      .hps_0_io_hps_io_sdio_inst_CMD     ( HPS_SD_CMD       ),
      .hps_0_io_hps_io_sdio_inst_D0      ( HPS_SD_DATA[0]     ),
      .hps_0_io_hps_io_sdio_inst_D1      ( HPS_SD_DATA[1]     ),
      .hps_0_io_hps_io_sdio_inst_CLK     ( HPS_SD_CLK      ),
      .hps_0_io_hps_io_sdio_inst_D2      ( HPS_SD_DATA[2]     ),
      .hps_0_io_hps_io_sdio_inst_D3      ( HPS_SD_DATA[3]     ),

//////////////////////////////////////  USB HOST  ///////////////
      .hps_0_io_hps_io_usb1_inst_D0      ( HPS_USB_DATA[0]    ),
      .hps_0_io_hps_io_usb1_inst_D1      ( HPS_USB_DATA[1]    ),
      .hps_0_io_hps_io_usb1_inst_D2      ( HPS_USB_DATA[2]    ),
      .hps_0_io_hps_io_usb1_inst_D3      ( HPS_USB_DATA[3]    ),
      .hps_0_io_hps_io_usb1_inst_D4      ( HPS_USB_DATA[4]    ),
      .hps_0_io_hps_io_usb1_inst_D5      ( HPS_USB_DATA[5]    ),
      .hps_0_io_hps_io_usb1_inst_D6      ( HPS_USB_DATA[6]    ),
      .hps_0_io_hps_io_usb1_inst_D7      ( HPS_USB_DATA[7]    ),
      .hps_0_io_hps_io_usb1_inst_CLK     ( HPS_USB_CLKOUT     ),
      .hps_0_io_hps_io_usb1_inst_STP     ( HPS_USB_STP        ),
      .hps_0_io_hps_io_usb1_inst_DIR     ( HPS_USB_DIR        ),
      .hps_0_io_hps_io_usb1_inst_NXT     ( HPS_USB_NXT        ),

////////////////////////////////////// UART 0 (Console)  ///////////
      .hps_0_io_hps_io_uart0_inst_RX     ( HPS_UART_RX        ),
      .hps_0_io_hps_io_uart0_inst_TX     ( HPS_UART_TX        ),


////////////////////////////////////////// HPS UART 1  ////////////
.hps_0_uart1_cts                    (),
.hps_0_uart1_dsr                    (),
```

```
.hps_0_uart1_dcd                    (),
.hps_0_uart1_ri                     (),
.hps_0_uart1_dtr                    (),
.hps_0_uart1_rts                    (),
.hps_0_uart1_out1_n                 (),
.hps_0_uart1_out2_n                 (),
.hps_0_uart1_rxd                    (uart1_rx),
.hps_0_uart1_txd                    (uart1_tx),

//////////////////////////////////////////////// I2C1  ////////////
.hps_0_i2c1_clk_clk             (scl1_o_e),
.hps_0_i2c1_scl_in_clk              (scl1_o),
.hps_0_i2c1_out_data                (sda1_o_e),
.hps_0_i2c1_sda                     (sda1_o),

//////////////////////////////////////////////// I2C3  ////////////
.hps_0_i2c3_scl_in_clk (scl3_o_e),
.hps_0_i2c3_clk_clk                 (scl3_o),
.hps_0_i2c3_out_data    (sda3_o_e),
.hps_0_i2c3_sda (sda3_o),

//////////////////////////////////////////////// CAN0  ////////////
.hps_0_can0_rxd                     (can0_rx),
.hps_0_can0_txd                     (can0_tx),

//////////////////////////////SPI0 Master  ///////////////////////
.hps_0_spim0_sclk_out_clk           (spi0_clk),
.hps_0_spim0_txd                    (spi0_mosi),
.hps_0_spim0_rxd                    (spi0_miso),
.hps_0_spim0_ss_in_n                (1'b1),
.hps_0_spim0_ssi_oe_n               (spim0_ssi_oe_n),
.hps_0_spim0_ss_0_n                 (spi0_ss_0_n),
.hps_0_spim0_ss_1_n                 (),
.hps_0_spim0_ss_2_n                 (),
.hps_0_spim0_ss_3_n                 (),

//////////////////////////// Analog Devices LTC2308    ////////////
.adc_ltc2308_external_interface_cs_n     (ADC_CONVST),
.adc_ltc2308_external_interface_sclk     (ADC_SCK),
.adc_ltc2308_external_interface_din      (ADC_SDI),
.adc_ltc2308_external_interface_dout     (ADC_SDO),

//////////////////////////////////////////  HPS LED & KEY  ///////
        .hps_0_io_hps_io_gpio_inst_GPIO53  ( HPS_LED),
        .hps_0_io_hps_io_gpio_inst_GPIO54  ( HPS_KEY),
```

```
////////////////////G-Sensor: I2C0 (Terasic Docu I2C1) ////////////
.hps_0_io_hps_io_i2c0_inst_SDA      (HPS_I2C1_SDAT),
.hps_0_io_hps_io_i2c0_inst_SCL      (HPS_I2C1_SCLK),

//////////////////////////onboard LEDs, Switches and Keys ///////////
.led_pio_external_connection_export (LED),
.pb_pio_external_connection_export  (KEY),
.sw_pio_external_connection_export  (SW),

////////////////////////////////// SPEED GPIO ///////////////////////
.speed_new_signal                           (GPI1GPIO[14]),

////////////////////////////////// READY GPIO ///////////////////////
.ready_new_signal                           (GPI1GPIO[28]),
////////////////////////////////// FAULT GPIO ///////////////////////

.fault_new_signal             (GPI1GPIO[18]), //FLT 1
        .fault_new_signal_1           (GPI1GPIO[20]), //FLT 2
        .fault_new_signal_2           (GPI1GPIO[22]),//FLT 3

////////////////////////////////// PWM 500KHz GPIO //////////////////
//    PWM control for Driver PS
    .pwm_new_signal               (GPI1GPIO[25]), //  .PWM_H
        .pwm_new_signal_1             (GPI1GPIO[23]),  //  .PWM_L
        .pwm_new_signal_2             (GPI1GPIO[24]),  //  .ENABLE
.pwm_new_signal_3             (GPI1GPIO[26]), //RST
////////////////////////////////// ADC Interface   //////////////////////////////////
//    Interface to connect with 3 ADC converters
.spi_1_new_signal             (GPI1GPIO[27]),    //ADC1 CS
        .spi_1_new_signal_1           (GPI1GPIO[29]),    //ADC1 MISO A
        .spi_1_new_signal_2           (GPI1GPIO[31]),    //ADC1 MISO B
        .spi_1_new_signal_3           (GPI1GPIO[33]),    //ADC1 MOSI
        .spi_1_new_signal_4           (GPI1GPIO[35]),    //ADC1 SCK
        .spi_2_new_signal             (GPI1GPIO[1]),     //ADC2 CS
        .spi_2_new_signal_1           (GPI1GPIO[3]),     //ADC2 MISO A
        .spi_2_new_signal_2           (GPI1GPIO[5]),     //ADC2 MISO B
        .spi_2_new_signal_3           (GPI1GPIO[7]),     //ADC2 MOSI
        .spi_2_new_signal_4           (GPI1GPIO[9]),     //ADC2 SCK

.spi_3_new_signal             (GPIOGPIO[17]),    //ADC3 CS
        .spi_3_new_signal_1           (GPIOGPIO[19]),    //ADC3 MISO A
        .spi_3_new_signal_2           (GPIOGPIO[21]),    //ADC3 MISO B
        .spi_3_new_signal_3           (GPIOGPIO[23]),    //ADC3 MOSI
        .spi_3_new_signal_4           (GPIOGPIO[25]),    //ADC3 SCK
```

```
        .spi_4_new_signal               (GPIOGPIO[7]),     //ADC4 CS
        .spi_4_new_signal_1             (GPIOGPIO[9]),     //ADC4 MISO A
        .spi_4_new_signal_2             (GPIOGPIO[11]),     //ADC4 MISO B
        .spi_4_new_signal_3             (GPIOGPIO[13]),     //ADC4 MOSI
        .spi_4_new_signal_4             (GPIOGPIO[15]),     //ADC4 SCK
////////////////////////////////// Sine PWM  GPIO ////////////////////////////////////

.spwm_new_signal                (GPI1GPIO[13]),    //PhaseH_A  PWM1
        .spwm_new_signal_1              (GPI1GPIO[21]),    //PhaseH_B  PWM5
        .spwm_new_signal_2              (GPI1GPIO[17]),    //PhaseH_C  PWM3
        .spwm_new_signal_3              (GPI1GPIO[11]),    //PhaseL_A  PWM2
        .spwm_new_signal_4              (GPI1GPIO[19]),    //PhaseL_B  PWM6
        .spwm_new_signal_5              (GPI1GPIO[15]),    //PhaseL_C  PWM4
        .spwm_new_signal_6              (GPI1GPIO[16]),    //SPWM ENABLE

////////////////////////////////// RELAYS GPIO ////////////////////////////////////

.relay_new_signal               (GPI1GPIO[30]),    //Relay A
        .relay_new_signal_1             (GPI1GPIO[32]),    //Relay B
        .relay_new_signal_2             (GPI1GPIO[34]),    //Relay C
////////////////////////////////// SPD SENSOR GPIO ////////////////////////////////
.spd_new_signal                 (GPI1GPIO[14]),

////////////////////////////////// HPS -> FPGA GPIO ////////////////////////////
//    32-Bit direct access registry between HPS and FPGA
  .hps_0_h2f_gp_gp_in (32'hACDCACDC), // FPGA to HPS -->
  .hps_0_h2f_gp_gp_out () // HPS to FPGA <--
);

////////////////////////////////////////// IO Buffer SPI 0 ////////////////////
// SPI0 -> CS
ALT_IOBUF spi0_ss_iobuf    (.i(spi0_ss_0_n), .oe(1'b1), .o(), .io(ARDUINO_IO[10]));
// SPI0 -> MOSI
ALT_IOBUF spi0_mosi_iobuf  (.i(spi0_mosi), .oe(1'b1), .o(), .io(ARDUINO_IO[11]));
// SPI0 -> MISO
ALT_IOBUF spi0_miso_iobuf  (.i(1'b0), .oe(1'b0), .o(spi0_miso), .io(ARDUINO_IO[12]));
// SPI0  -> CLK
ALT_IOBUF spi0_clk_iobuf   (.i(spi0_clk), .oe(1'b1), .o(), .io(ARDUINO_IO[13]));

////////////////////////////////////////// IO Buffer I2C 1 and 3 //////////////
// I2C1 -> SCL
ALT_IOBUF i2c1_scl_iobuf   (.i(1'b0),.oe(scl1_o_e),.o(scl1_o),.io(ARDUINO_IO[15]));
// I2C1 -> SDA
ALT_IOBUF i2c1_sda_iobuf   (.i(1'b0),.oe(sda1_o_e),.o(sda1_o),.io(ARDUINO_IO[14]));
```

```
// I2C3 -> SCL
ALT_IOBUF i2c3_scl_iobuf   (.i(1'b0),.oe(scl3_o_e),.o(scl3_o),.io(ARDUINO_IO[3]));
// I2C3 -> SDA
ALT_IOBUF i2c3_sda_iobuf   (.i(1'b0),.oe(sda3_o_e),.o(sda3_o),.io(ARDUINO_IO[2]));

////////////////////////////////////// IO Buffer UART1  /////////////////
// UART1 -> RX
ALT_IOBUF uart1_rx_iobuf (.i(1'b0), .oe(1'b0), .o(uart1_rx), .io(ARDUINO_IO[1]));
   // UART1 -> TX
ALT_IOBUF uart1_tx_iobuf (.i(uart1_tx), .oe(1'b1), .o(), .io(ARDUINO_IO[0]));

////////////////////////////////////// IO Buffer CAN0  /////////////////
// CAN0 -> RX
ALT_IOBUF can0_rx_iobuf (.i(1'b0), .oe(1'b0), .o(can0_rx), .io(ARDUINO_IO[9]));
   // CAN-> TX
ALT_IOBUF can0_tx_iobuf (.i(can0_tx), .oe(1'b1), .o(), .io(ARDUINO_IO[8]));

endmodule
```