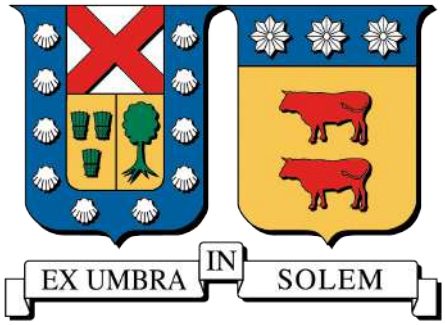
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE



IMPLEMENTACIÓN EN FPGA DE CONTROL PREDICTIVO BASADO EN MODELOS EN UN CONVERTIDOR DC/AC CONECTADO A UN FILTRO LC EN DERS.

JUAN DAVID ESCÁRATE MUÑOZ

MEMORIA DE TITULACIÓN Y TESIS DE GRADO PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA ELECTRÓNICA Y EL TÍTULO DE INGENIERO CIVIL ELECTRÓNICO

PROFESOR GUÍA	:	GONZALO CARVAJAL
PROFESOR CORREFERENTE	:	CÉSAR SILVA

11 DE ABRIL DE 2023

Agradecimientos

Quiero agradecer al proyecto Fondecyt Regular 1211676, y a los profesores César Silva, Gonzalo Carvajal y Juan Agüero, por su ayuda en este proceso. Espero que el desarrollo de esta tesis sea de gran utilidad para el progreso del grupo de investigación.

En un ámbito personal, quiero destacar el cariño y esfuerzo de mi familia en mis años universitarios, donde me brindaron un apoyo que me permitió llevar a cabo cada uno de los logros que me propuse.

Quiero dedicarle esta tesis a mis padres, mi hermana, mi tío y mis dos grandes amigos, Julio y Rojo.

Resumen

En este trabajo se presenta una implementación en hardware para la etapa de control de una aplicación de electrónica de potencia, la cual consta de una fuente de energía DC, un inversor, un filtro LC y una carga trifásica balanceada. El controlador consiste en un enfoque *Continuous Control Set Model Predictive Control*, el cual se basa en calcular la actuación óptima de manera *online* en base a las mediciones de voltaje y corriente de salida, para regular la tensión de salida en magnitud y frecuencia. Debido a que existe una restricción de tiempo real definida por el tiempo de muestreo, se diseña una implementación en *Field Programmable Gate Array* (FPGA) para obtener una arquitectura determinista y eficiente.

Específicamente, se utiliza una simulación de MATLAB/Simulink con el solver *quadprog* para analizar el desempeño del controlador propuesto al utilizar el *solver Alternating Direction Method of Multipliers* (ADMM), utilizando un caso de estudio con seis configuraciones, variando la cantidad de iteraciones del *solver* ADMM. A través del análisis en simulación se determina que el valor de *Total Harmonic Distortion* para el voltaje de salida es similar utilizando *quadprog* o ADMM. Al no existir una diferencia relevante entre los voltajes de salida en estado estacionario, se estudia el comportamiento en estado transiente, donde ADMM presenta sobrecorriente, la cual se visualiza al realizar cambio de carga exigiendo el valor nominal de corriente a la fuente de energía. En esta simulación se considera un tiempo de muestreo de 200 µs, imponiendo un requisito de tiempo real para la implementación del controlador. Para este propósito se desarrolla una etapa de diseño de hardware utilizando FPGAs a través de las herramientas Vitis HLS, Vivado y PYNQ, de AMD Xilinx. En este trabajo se utiliza la tarjeta de desarrollo AMD Xilinx ZCU104, la cual es un *MultiProcessor System on Chip* (MPSoC) que contiene: CPU *multicore*, FPGA, periféricos, entre otros.

Los resultados de la implementación de las seis configuraciones del *solver* ADMM en FPGA demuestran la capacidad de computar la solución óptima al problema de optimización del controlador, cumpliendo con la restricción temporal de 200 µs. Sin embargo, existe una etapa de transmisión que agrega un *overhead*, por lo que se crea una implementación del controlador que incluye tanto el procesamiento de mediciones como la resolución del problema de optimización. En esta segunda implementación se logra computar una solución óptima dentro de la restricción de tiempo real para las tres configuraciones de ADMM diseñadas. Adicionalmente se reportan los tiempos de ejecución en CPU para comparar la velocidad de cómputo, logrando una aceleración de 20x para la implementación de ADMM y 33x para la implementación del controlador.

El flujo de diseño y los resultados proveen evidencia experimental del potencial de FPGAs para la implementación de esquemas avanzados de control para la operación de inversores con fuentes DC y cargas trifásicas balanceadas.

Índice general

Índice de tablas

Índice de figuras

Códigos

1 Introducción

1.1. Motivación y contexto

La creciente utilización de energías renovables en el sistema eléctrico, diversidad de cargas y baja inercia de sistemas de generación energética distribuidos, introduce desafíos para la operación de sistemas eléctricos convencionales. El comportamiento impredecible de energías renovables en recursos de energía distribuidos (DERs) y la variabilidad en la demanda de potencia introducen requerimientos técnicos relacionados a mantener la estabilidad de la magnitud y frecuencia del voltaje del sistema, protegiéndolo contra posibles sobrecargas transitorias. Por lo anterior, se ha vuelto imperativo construir arquitecturas de sistemas eléctricos distribuidos para proveer eficiencia, fiabilidad, escalabilidad y resiliencia a la red, aspectos que se obtienen a través del uso de micro-redes modulares y escalables [1]. Las micro-redes son sistemas de energía autosuficientes compuestos por DERs y cargas controlables, pudiendo desconectarse de la red eléctrica principal y operar de manera autónoma. Ejemplos típicos de DERs incluyen baterías de respaldo, generadores de emergencia, paneles solares, etc. Los DERs permiten generación y consumo local de electricidad, además de proveer servicios auxiliares para fortalecer la resiliencia de la red y mitigar perturbaciones [2, 3].

Un DER contiene elementos de corriente directa (DC) (por ejemplo, paneles solares) conectados a una red de corriente alterna (AC) a través de un convertidor de potencia DC/AC que permite el flujo de potencia entre un sistema DC y uno AC. En la topología de un convertidor de potencia típicamente se encuentran dispositivos electrónicos como diodos y transistores, los cuales se diseñan para operar en valores nominales de voltaje, corriente, temperatura, entre otros. Además, es común incluir un filtro LC (inductivo-capacitivo) en la salida para atenuar armónicas de alta frecuencia introducidas en el bus de energía [4]. En el sistema conformado por DERs junto con filtro LC de salida, es necesario el uso de un esquema de control con el fin de regular voltaje y frecuencia de salida que van conectados a una red estable. Al mismo tiempo, es necesario que las acciones de control mantengan las variables internas dentro de los rangos nominales de operación para evitar daños a los dispositivos y pérdidas por mal funcionamiento.

Una forma de incorporar las restricciones de actuación para convertidores de potencia es utilizar esquemas de control del tipo *Model Predictive Control* (MPC). MPC utiliza un modelo dinámico del sistema eléctrico para predecir el comportamiento futuro de estados y salidas del sistema ante una secuencia de entradas de control, permitiendo llevar las salidas a un valor deseado respetando restricciones en variables internas a través de un modelo de optimización. El modelo de optimización permite lograr una actuación óptima utilizando información pasada del comportamiento del sistema y la estimación del comportamiento futuro dada por el modelo

dinámico del sistema. En [5], los autores concluyen que MPC desempeñará un papel vital para que la próxima generación de convertidores de potencia y accionamientos eléctricos funcionen de manera eficiente. Además, MPC se ha aplicado de manera satisfactoria a dispositivos de electrónica de potencia en formulaciones de *Finite Control Set* (FCS) y *Continuous Control Set* (CCS) [6, 7].

El enfoque FCS define un número finito de estados de conmutación que un inversor de potencia puede adoptar, simplificando el problema de optimización al considerar sólo estos posibles estados para predecir el comportamiento del convertidor. Adicionalmente, el esquema de FCS no presenta una etapa de Modulación por Ancho de Pulso (PWM). Por otro lado, CCS requiere de una etapa de PWM de frecuencia fija que comanda la conmutación de los transistores presentes en la topología del convertidor. En este contexto, CCS-MPC tiene dos enfoques: uno de ellos se basa en computar de manera *online* la actuación (conocido como *Generalized Predictive Control* o GPC), mientras que el otro guarda de manera offline los posibles resultados en un *look up table* (conocido como *Explicit* MPC o EMPC) [6, 8]. En este trabajo se utiliza el enfoque *online* donde se requiere resolver un modelo de optimización continuamente considerando perturbaciones y restricciones de operación para lograr una actuación óptima. Comparativamente, la principal ventaja de CCS-MPC sobre FCS-MPC es la utilización de PWM de frecuencia fija, resultando en que CCS-MPC presenta una mejor distorsión armónica total (THD) [9] y simplifica el diseño de la arquitectura de control [10].

Trabajos recientes en la literatura académica han mostrado las ventajas del uso de esquemas CCS-MPC para controlar la magnitud y frecuencia del voltaje en DERs [11, 12]. No obstante, estos trabajos únicamente llevan a cabo simulaciones para evaluar el algoritmo de control desde una perspectiva funcional, sin considerar restricciones temporales ni requerimientos computacionales para su implementación efectiva. El enfoque de CCS-MPC presentado en este trabajo requiere resolver un problema de *Quadratic Programming* (QP) para encontrar las entradas óptimas de control en cada intervalo de muestreo, cuya duración viene definida por el período de la etapa de modulación. En [12], los autores indican un objetivo de control asociado a la regulación en magnitud y frecuencia del voltaje de salida de un sistema similar al presentado en esta tesis. Además, el sistema presentado en su trabajo indica un período de muestreo de 200 µs, imponiendo requerimientos de tiempo real para el cómputo en cada iteración de control que son difíciles de alcanzar con procesadores embebidos de propósito general. Debido a estos requerimientos computacionales, la literatura reciente muestra un creciente interés en utilizar *Field-Programmable Gate Arrays* (FPGAs) para la implementación eficiente de lazos MPC con garantías temporales mediante arquitecturas especializadas [13–15]. A pesar de que la literatura ilustra los beneficios de implementar MPC en FPGAs, las técnicas propuestas en estas referencias corresponden a ejemplos con sistemas genéricos simplificados (sistema masa resorte, lasso, entre otros) que cumplen un propósito didáctico y demostrativos del concepto. Estos trabajos suelen enfocarse en los aspectos conceptuales y algorítmicos, sin entregar muchos detalles sobre la implementación final en las plataformas de cómputo especializadas, lo cual sumado a la inherente complejidad del diseño e implementación en FPGAs, imposibilita la reproducción de los resultados y su extensión para aplicaciones específicas dentro de la electrónica de potencia cuyos requerimientos de cómputo de tiempo real son más exigentes que en los sistemas típicamente usados como referencia.

En esta tesis se reporta un desarrollo técnico y práctico del uso de CCS-MPC en sistemas de potencia, por medio del diseño, implementación y evaluación de una arquitectura de cómputo especializada basada en FPGA para regular la magnitud y frecuencia del voltaje de un filtro LC

conectado un convertidor DC/AC que es alimentado a través de DERs. Además de demostrar la factibilidad de implementación, se busca aprovechar el determinismo y paralelismo presente en microarquitecturas diseñadas en FPGA para acelerar la ejecución de un bucle CCS-MPC adaptado al requisito de tiempo real de la aplicación, la cual corresponde a un tiempo de ejecución del algoritmo de control no mayor a 200 µs.

1.2. Planteamiento del problema

El cómputo online de un modelo de optimización tipo QP para un lazo de control CCS-MPC exige que la solución se obtenga dentro del tiempo de muestreo definido por la etapa de modulación, el cual se denomina a lo largo de este documento como "requerimiento de tiempo real". Dado que los sistemas de electrónica de potencia requieren de altas frecuencias de conmutación para obtener señales sinusoidales con bajo THD, se busca una implementación computacional altamente eficiente para lograr estos requerimientos de tiempo que se encuentran en torno a los cientos de microsegundos. Además, en términos de cómputo, se ha estimado en [12] a través de simulaciones en MATLAB que el cuello de botella del controlador se encuentra en la obtención de la solución óptima del modelo de optimización planteado.

Para la solución de un modelo de optimización QP existen varias alternativas presentes en la literatura [15]. Sin embargo, muchas de ellas requieren de estructuras computacionales que se adecuen a las operaciones matemáticas que forman parte del proceso de resolución. Según se muestra en [15], el algoritmo *Alternating Direction Method of Multipliers* (ADMM) se presenta como un *solver* de primer orden que requiere de álgebra matriz-vector basada en sumas y multiplicaciones, capaz de otorgar una solución dentro del rango de cientos de microsegundos para las aplicaciones que desarrolla cada autor, mostrándolo como un candidato ideal para soluciones embebidas.

Para el desarrollo de arquitecturas especializadas en sistemas embebidos existen soluciones comerciales que incluyen el uso de FPGAs para implementaciones computacionalmente eficientes. Sin embargo, para aprovechar la disponibilidad de recursos, el determinismo y un control fino de la plataforma de desarrollo, se prefiere el uso de FPGAs como aceleradoras de cómputo autónomas. A pesar de que, las herramientas para diseño en FPGAs se basan en el uso de *Hardware Description Language* (HDL), existen alternativas de alto nivel para el desarrollo de arquitecturas de hardware. En particular, las herramientas de *High Level Synthesis* (HLS), se presentan como entornos de trabajo que permiten el uso de lenguajes de alto nivel como C/C++ para el diseño de arquitecturas de hardware. Esto permite que los desarrolladores eviten el uso de diseño RTL de bajo nivel de manera directa, logrando implementaciones altamente eficientes descritas en lenguajes de alto nivel.

Considerando la información descrita, se plantea formalmente el problema de la siguiente forma: *dado un lazo de control CCS-MPC de un convertidor DC/AC con filtro LC de salida que permite regular voltaje y frecuencia de salida, considerando restricciones de corriente, es posible computar la solución del problema de optimización del controlador a través de una implementación de ADMM en FPGA, utilizando herramientas de diseño de alto nivel, alcanzando un tiempo de ejecución menor a 200 µs.*

1.3. Alcances y contribuciones

Este trabajo se enfoca en el estudio, implementación y evaluación del algoritmo ADMM en FPGAs como *solver* de optimización para un control tipo CCS-MPC, con el fin de regular el voltaje de salida de un convertidor de potencia DC/AC conectado a un filtro LC. Dicho esto, para este trabajo de tesis se consideraron los siguientes alcances:

A1 El trabajo utiliza como referencia la aplicación presentada en [12]. Dado que las condiciones planteadas en la aplicación de referencia indican un tiempo de muestreo de 200 µs, se define este valor como requerimiento de tiempo real. Además, este documento es complementario al análisis teórico presentado en la tesis de Magíster de Reinier Lopez [12].

A2 Tras la revisión de literatura, se decide utilizar ADMM por dos razones: el uso de un método primal-dual ocasiona que la implementación sea fácilmente paralelizable [16], y las operaciones matemáticas iterativas son principalmente adiciones y multiplicaciones vector-matriz transformándolo en candidato ideal para soluciones en sistemas embebidos [15, 17–21].

A3 Para la evaluación experimental se utilizará la plataforma de desarrollo ZCU104 del fabricante AMD Xilinx junto con las herramientas de diseño Vivado, Vitis HLS. La tarjeta es un *MultiProcessor System on Chip* (MPSoC), por lo que contiene CPU, FPGA, periféricos, entre otros, permitiendo un análisis para ejecución en CPU y FPGA en la misma plataforma de desarrollo.

A4 El enfoque de este trabajo es reportar los tiempos de ejecución, por lo que el sistema de referencia es simulado. Esto quiere decir que no se presenta una configuración con elementos de electrónica de potencia, si no una representación simulada de las señales dinámicas de la aplicación objetivo.

Basado en los resultados obtenidos, se identifica la siguiente principal contribución:

C1 Se entrega evidencia del uso de herramientas de HLS para el desarrollo de un esquema de control tipo CCS-MPC con cálculos *online* utilizando una plataforma de desarrollo ZCU104. Además, este trabajo es complementario a un repositorio[1] que contiene información y códigos para replicar los resultados.

Los resultados del trabajo presentado buscan abrir una línea investigativa enfocada en la exploración de optimizaciones utilizando las herramientas de alto nivel y una implementación de un lazo de control predictivo, agregando alternativamente un observador de control a futuro.

1.4. Organización del informe

El contenido de esta tesis está organizado de la siguiente forma:

- Capitulo 2 presenta el sistema de referencia y los fundamentos teóricos de cada elemento que lo conforma, así como un desarrollo matemático del algoritmo de optimización escogido.

[1] Los códigos de bajo nivel se encuentran con acceso restringido. Para mayor información contactar a los profesores a cargo del grupo de investigación: César Silva, Gonzalo Carvajal o Juan Agüero, del Departamento de Electrónica de la Universidad Técnica Federico Santa María.

- Capitulo 3 presenta la simulación del sistema de referencia en MATLAB/Simulink donde se muestra el desempeño del lazo de control al variar las iteraciones del solver ADMM. Además, se detallan las decisiones de diseño y se definen los requerimientos de tiempo de la aplicación.
- Capitulo 4 muestra el flujo de diseño y explica la utilización de herramientas de diseño de hardware desarrolladas por Xilinx, las cuales son: Vitis HLS, Vivado, y PYNQ.
- Capitulo 5 presenta los resultados de este trabajo donde se comparan los tiempos de ejecución entre FPGA y CPU para dos enfoques de implementación utilizando la plataforma de desarrollo ZCU104.
- Capitulo 6 concluye esta tesis con un resumen del trabajo realizado y los resultados obtenidos, así como enfoques a futuro para complementar el trabajo realizado.

2 Antecedentes conceptuales y técnicos

El sistema de referencia utilizado en esta tesis consta de un convertidor DC/AC conectado a un sistema de alimentación de corriente continua, el cual se presenta como un DER para efectos de este trabajo, mientras que la salida es una carga trifásica AC. Además, se utiliza un filtro inductivo-capacitivo entre la salida del convertidor y la carga para eliminar las armónicas de alta frecuencia, ocasionando que la señal que reciba la carga presente un contenido armónico con menos distorsión. Cabe señalar que el diseño de este sistema es una variación del trabajo realizado por Reinier Lopez [12], utilizado con autorización y apoyo del autor.

Este capítulo tiene como finalidad introducir la aplicación objetivo y explicar cada uno de los componentes del sistema de referencia. De esta forma, se presenta el contexto del esquema de control CCS-MPC donde se desea implementar el algoritmo de optimización ADMM.

2.1. Diagrama de sistema de refencia y esquema de control

El sistema de referencia (ver Figura 2.1a) contiene una fuente de energía tipo DC conectada a un convertidor DC/AC para suplir una carga con conexión tipo estrella. De esta forma, la representación de tanto el filtro LC como la carga se deben ampliar a cada una de las tres fases. Por otra parte, el esquema de control (ver Figura 2.1b) mide el voltaje y la corriente de salida, para que junto a un voltaje de referencia en coordenadas dq, se procese una actuación óptima en el bloque CCS-MPC. Los componentes del sistema de referencia y el esquema de control se explican en detalle en el desarrollo de este capítulo.

2.2. Convertidor trifásico AC/DC

En un sistema eléctrico se pueden encontrar principalmente dos categorías de componentes eléctricas, AC y DC, que se diferencian según el tipo de corriente que generan o consumen. Un subsistema AC contiene señales que tienen un comportamiento periódico, las cuales típicamente se presentan en sistemas eléctricos como señales sinusoidales. Adicionalmente, los subsistemas AC suelen utilizarse para describir etapas de generación y/o transmisión en sistemas de alta o media potencia de manera monofásica o trifásica. Por otra parte, los subsistemas DC se pueden presentar de la siguiente forma: cargas eléctricas, sistemas de generación que utilizan paneles fotovoltaicos o generadores DC, o almacenamiento de energía en baterías. Para el flujo de potencia entre sistemas eléctricos AC y DC, se utiliza convertidores AC/DC o DC/AC, dependiendo de la dirección en que fluye la potencia, los cuales son ampliamente reconocidos

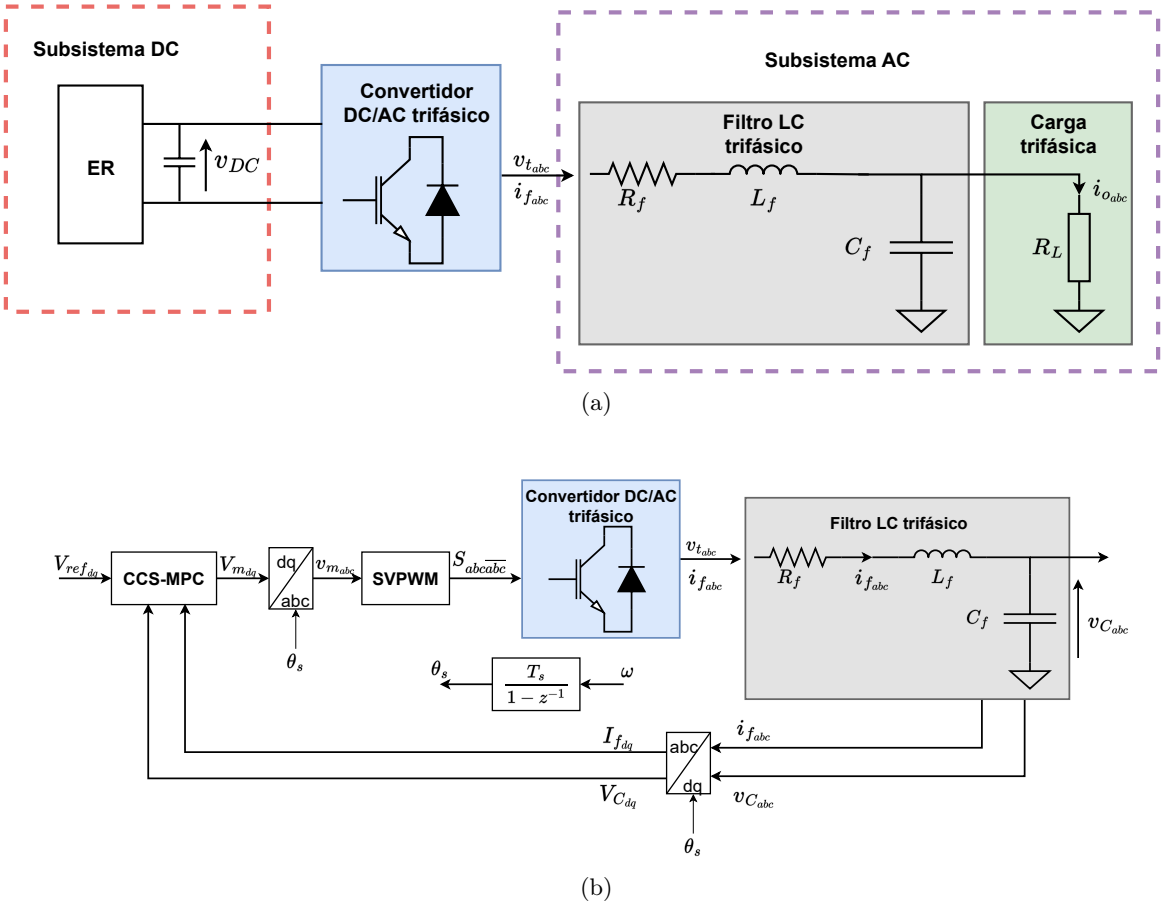


Figura 2.1: (a) Sistema de referencia junto con su (b) esquema de control.

en la literatura como "inversores" [22].

En el sistema de referencia utilizado en este trabajo (ver Figura 2.1a), se define una etapa AC trifásica, por lo que todos los dispositivos electrónicos que forman parte del subsistema AC deben seguir una topología trifásica. En la topología de inversor trifásico utilizada en este trabajo (ver Figura 2.2), se requiere de tres señales de control S_a, S_b y S_c, y sus correspondientes valores negados S_{an}, S_{bn} y S_{cn}. El patrón de conmutación de los seis transistores, que afecta la forma de onda de las señales v_a, v_b y v_c, proviene de una etapa de modulación que se encarga de transformar una forma de onda continua en señales de disparo. Con ello, el espectro en frecuencia de la onda resultante contiene gran parte de la energía en la frecuencia de interés, frecuencia que es predefinida según los requerimientos del subsistema AC. A pesar de la existencia de varias implementaciones en la literatura de la etapa de modulación [23], en este trabajo utilizaremos la técnica de *Space Vector* PWM (SVPWM), la cual es ampliamente utilizada en sistemas trifásicos [24].

2.3. *Space Vector Pulse Width Modulation* (SVPWM)

La modulación por ancho de pulso es una técnica que utiliza pulsos de ancho variable para representar la amplitud de una señal análoga. En término eléctricos, PWM permite controlar la potencia media entregada a una carga por medio de la variación del ciclo de trabajo de

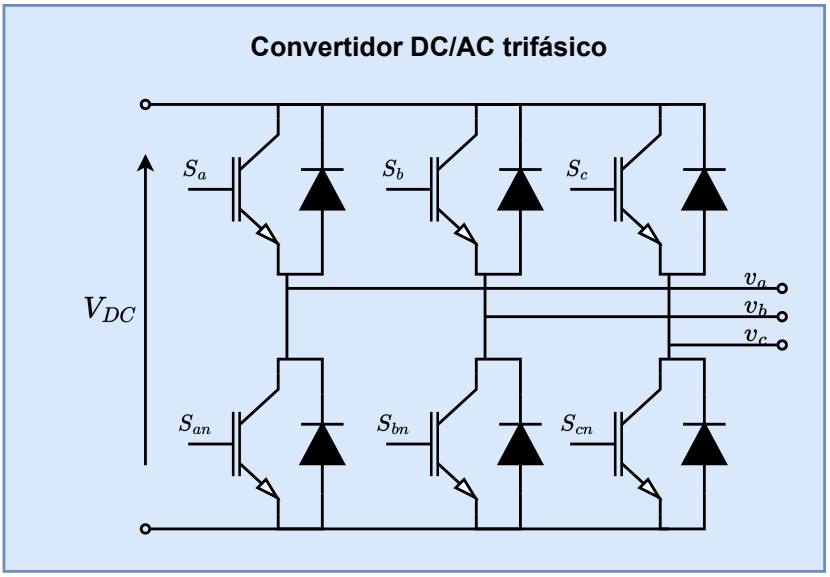


Figura 2.2: Topología convertidor DC/AC

trenes de pulsos. En particular, los voltajes de salida del sistema de referencia (v_a, v_b y v_c en Figura 2.2) se modulan a través de una técnica llamada SVPWM. Para la implementación de SVPWM, existe una método que es equivalente y se denomina inyección de min-max [25]. Para este trabajo se utiliza el método min-max, el cual presenta un esquema como el mostrado en la Figura 2.3. De manera particular, las señales de referencia v_{ma}, v_{mb} y v_{mc} provienen directamente del bloque de control CCS-MPC de la Figura 2.1a. Adicionalmente, la frecuencia de la portadora (señal triangular de la Figura 2.3) se define de manera previa teniendo en cuenta tanto que debe ser mayor tanto a la frecuencia de las señales de referencia como a la frecuencia de muestreo del sistema.

La idea detrás de este método es sustraer una señal con componente de tercer armónico a la señal de referencia. La señal con componente de tercer armónico $v_{min-max}$ se obtiene según la siguiente relación

$$v_{min-max} = \frac{\text{mín}(v_{ma}, v_{mb}, v_{mc}) + \text{máx}(v_{ma}, v_{mb}, v_{mc})}{2}, \tag{2.1}$$

donde v_{ma}, v_{mb} y v_{mc} corresponden a las señales de referencia para las fases a, b y c respectivamente.

De esta forma, se compara la señal $v^*_{mj} = v_{mj} - v_{min-max}$, donde $j \in \{a, b, c\}$, con la portadora a través de un amplificador operacional, logrando una salida binaria para conmutar los transistores.

Por ejemplo, para un sistema eléctrico trifásico, las señales de referencia se pueden definir como:

$$\begin{aligned} v_{ma} &= \hat{V} sin(2\pi f) \\ v_{mb} &= \hat{V} sin(2\pi f - 2\pi/3) \\ v_{mc} &= \hat{V} sin(2\pi f + 2\pi/3), \end{aligned}$$

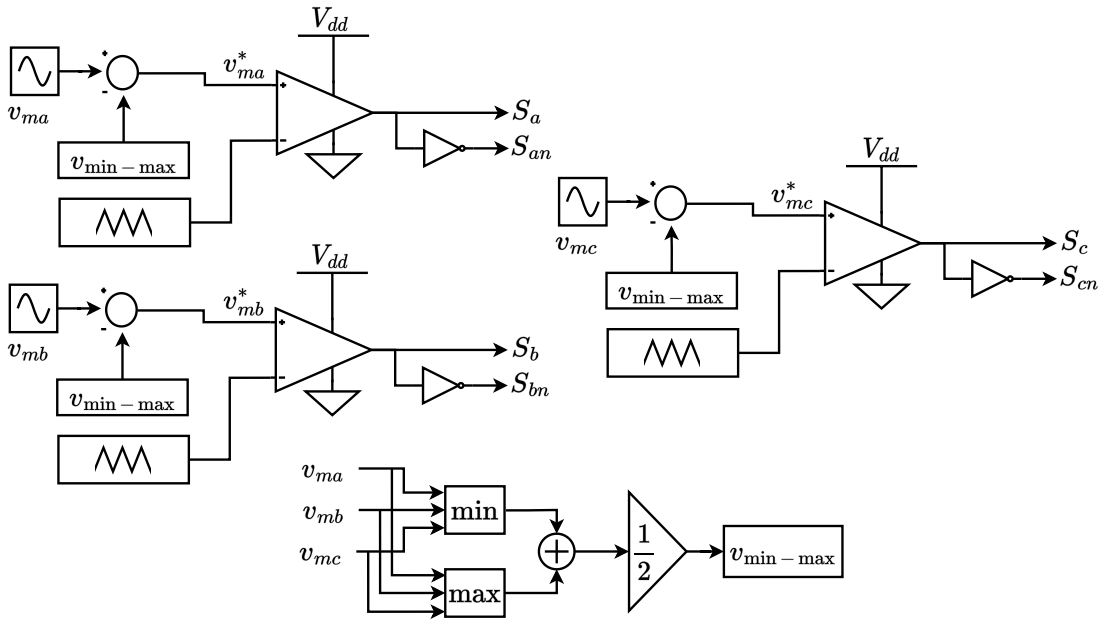


Figura 2.3: Esquema de modulación SVPWM con inyección min-max para señales de referencia sinusoidales.

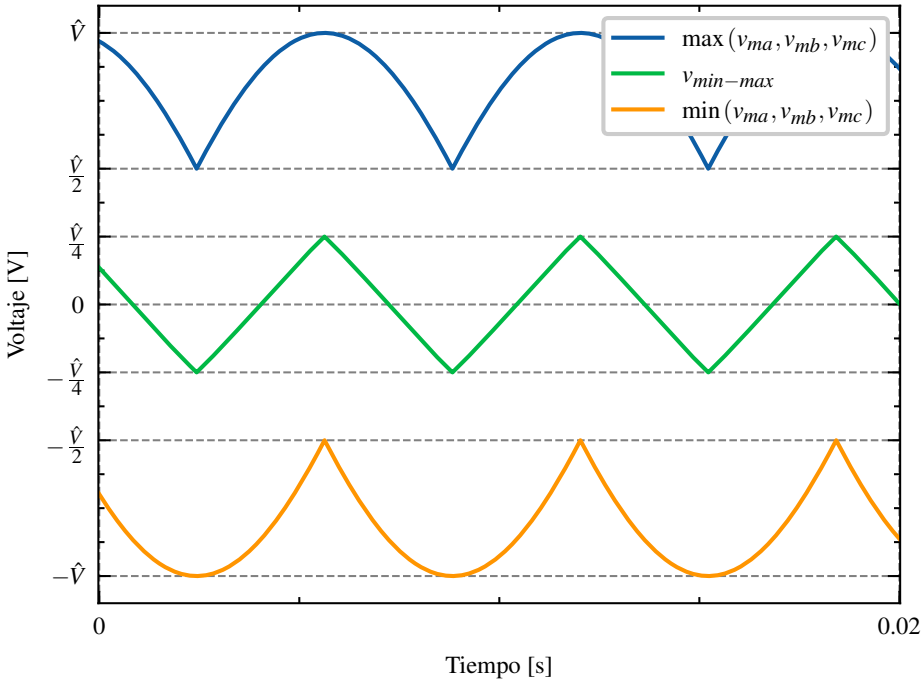
donde $\hat{V}$ representa la amplitud y f la frecuencia de la señal de referencia.

Con la finalidad de mostrar las señales internas del esquema de modulación min-max para SVPWM, se crea un ejemplo con una amplitud arbitraria ($\hat{V}$) y una frecuencia de 50 Hz. En la Figura 2.4a se ilustran las señales que representan el máximo entre v_{ma}, v_{mb} y v_{mc} de color azul, y el mínimo entre v_{ma}, v_{mb} y v_{mc} de color naranjo. El promedio entre estas dos señales ($v_{min-max}$) se puede ver como una señal triangular de color verde con valor medio cero y amplitud $\hat{V}/4$, cuya frecuencia equivale a tres veces la frecuencia de la señal de referencia, es decir, 150 Hz. Al sustraer $v_{min-max}$ de la referencia de la fase A se tiene una forma de onda como la que se muestra de color azul (v_{ma^*}) en la Figura 2.4b, la cual se encuentra normalizada con respecto a $\hat{V}$. Finalmente, se compara la portadora (verde) de Figura 2.4b con v_{ma^*} para la fase a, obteniendo como resultado S_a (naranjo), el cual indica las conmutaciones para dos de los seis transistores de la topología del inversor presentada en la Figura 2.2. Esto último se debe a que S_a y S_{an} son señales opuestas. Es importante mencionar que la frecuencia de la portadora debe ser mayor a frecuencia de la señal de referencia para una modulación correcta. En general, se diseña una portadora al menos diez veces más rápida que la referencia.

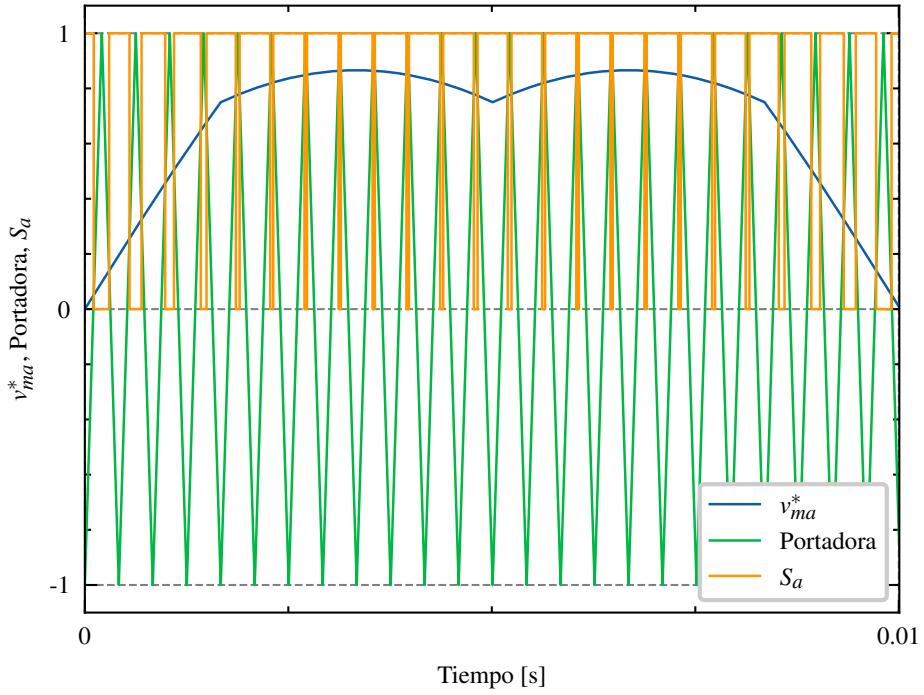
La forma de onda del voltaje de salida se define según el patrón de conmutación en los transistores del inversor. Sin embargo, dado que el control se diseña en coordenadas dq, se requiere que la salida del controlador sea transformada desde coordenadas dq a coordenadas trifásicas para poder ser moduladas de manera correcta.

2.4. Transformada dq

La transformada dq, o denominada transformada de Park, se utiliza para cambiar el marco de referencia de un sistema eléctrico trifásico a uno bifásico. La principal ventaja es que, al tener una representación en 2D, se pueden utilizar números complejos y, por consiguiente, diseñar



(a)



(b)

Figura 2.4: (a) Señales de máximo y mínimo de las tres fases y $v_{min-max}$ (2.1). (b) Señales relevantes para operación del convertidor de potencia.

de manera independiente el esquema de control en el eje real y en el eje imaginario. Con esta transformación se logra un esquema que consta de dos valores de referencia, uno en el eje d (real) y el otro en el eje q (imaginario).

Antes de comenzar el desarrollo de la transformación, se definen los vectores de entrada para un sistema eléctrico trifásico según

$$\begin{aligned} V &= (V_a, V_b, V_c) \\ V_a &= \hat{V} sin(2\pi f) \\ V_b &= \hat{V} sin(2\pi f - 2\pi/3) \\ V_c &= \hat{V} sin(2\pi f + 2\pi/3), \end{aligned}$$

donde $\hat{V}$ representa la amplitud y f la frecuencia de la señal, mientras que a, b y c representan las tres fases del sistema eléctrico.

Primero, se define la transformada de Clarke como

$$T_C = \frac{2}{3} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix}.$$

Al utilizar esta transformada, se trabaja en un plano conocido en la literatura como coordenadas $\alpha\beta$ [26]. De esta representación, se obtiene la siguiente relación

$$\begin{aligned} \begin{bmatrix} V_\alpha \\ V_\beta \\ V_\gamma \end{bmatrix} &= \frac{2}{3} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} \\ V_{\alpha\beta} &= T_C \cdot V. \end{aligned}$$

Comúnmente, la carga se encuentra balanceada, por lo que $I_\gamma = I_a + I_b + I_c = 0$, lo que implica que $V_\gamma = 0$. Es por esto que, además de "Transformada de Clarke", se le conoce como "Transformada $\alpha\beta$" o "Transformada $\alpha\beta 0$".

Luego, se realiza una rotación de coordenadas según

$$T_R = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

logrando un vector en coordenadas dq definido como

$$\begin{aligned} \begin{bmatrix} V_d \\ V_q \\ V_0 \end{bmatrix} &= \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_\alpha \\ V_\beta \\ V_\gamma \end{bmatrix} \\ V_{dq} &= T_R \cdot V_{\alpha\beta} \\ V_{dq} &= T_R \cdot T_C \cdot V. \end{aligned} \tag{2.2}$$

Finalmente, la transformada de Park (T_P) se define como

$$T_P = T_R \cdot T_C = \frac{2}{3}\begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

$$T_P = \frac{2}{3}\begin{bmatrix} \cos(\theta) & \frac{\sqrt{3}}{2}\sin(\theta) - \frac{1}{2}\cos(\theta) & -\frac{1}{2}\cos(\theta) - \frac{\sqrt{3}}{2}\sin(\theta) \\ -\sin(\theta) & \frac{1}{2}\sin(\theta) + \frac{\sqrt{3}}{2}\cos(\theta) & \frac{1}{2}\sin(\theta) - \frac{\sqrt{3}}{2}\cos(\theta) \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

$$T_P = \frac{2}{3}\begin{bmatrix} \cos(\theta) & \cos\left(\theta - \frac{2\pi}{3}\right) & \cos\left(\theta + \frac{2\pi}{3}\right) \\ -\sin(\theta) & -\sin\left(\theta - \frac{2\pi}{3}\right) & -\sin\left(\theta + \frac{2\pi}{3}\right) \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix}. \tag{2.3}$$

De esta forma, al reemplazar (2.3) en (2.2) se obtiene lo siguiente

$$V_{dq} = T_P \cdot V. \tag{2.4}$$

Para la transformada de Park se requiere el valor del ángulo de rotación de los ejes (θ), el cual se obtiene mediante una etapa de integración de la frecuencia eléctrica que viene dada según la siguiente ecuación

$$\frac{d\theta(t)}{dt} = \omega$$
$$\theta(t) = \int_0^t \omega + \theta(0).$$

El vector resultante V_{dq} contiene dos valores con información del sistema, V_d y V_q, los cuales se suelen escribir de manera compleja $V_{dq} = V_d + jV_q$ para facilitar el manejo algebraico y el diseño del controlador ya que ambos valores son ortogonales.

El desarrollo algebraico presentado en esta sección se muestra para la transformación de coordenadas trifásicas a dq, para transformar desde coordenadas dq a trifásicas se utiliza la siguiente relación

$$V = T_P^{-1} \cdot V_{dq}$$
$$T_P^{-1} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 1 \\ \cos\left(\theta - \frac{2\pi}{3}\right) & -\sin\left(\theta - \frac{2\pi}{3}\right) & 1 \\ \cos\left(\theta + \frac{2\pi}{3}\right) & -\sin\left(\theta + \frac{2\pi}{3}\right) & 1 \end{bmatrix}.$$

Una vez empleada la transformada dq para las mediciones, se definen los valores de referencia para d y q, para así diseñar un controlador que sea capaz de seguir la referencia en estado estacionario.

2.5. *Continuous Control Set Model Predictive Control* (CCS-MPC)

MPC es una técnica de control basada en el uso de modelos para predecir y optimizar la actuación a través de señales controlables sobre un sistema. Para esta finalidad, se requiere un modelo dinámico que describa el comportamiento del sistema y un modelo de optimización. El modelo dinámico se define mediante una representación de espacio estado, el cual provee información acerca de la dinámica del sistema utilizando información pasada y entradas actuales. Por otra parte, la etapa de optimización se describe mediante un modelo tipo QP, o bien, a través de costos cuadráticos, el cual considera las restricciones de actuación para obtener una solución óptima de las entradas controlables.

Actualmente existen implementaciones de MPC a dispositivos de electrónica de potencia en sus dos formulaciones, *Finite Control Set* (FCS) y *Continuous Control Set* (CCS) [7]. En este trabajo se emplea un controlador de tipo CCS-MPC con un enfoque asociado al cómputo *online* de un problema de optimización. Para comprender la teoría detrás del controlador propuesto, se desarrolla la obtención del modelo de optimización en base al modelo espacio-estado del sistema.

2.5.1. Modelo espacio-estado

Un modelo espacio-estado discreto lineal e invariante en el tiempo se define según

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k + Du_k, \end{aligned} \tag{2.5}$$

donde $x_k \in \mathbb{R}^n$ representa el valor de los n estados del sistema en el instante k, $y_k \in \mathbb{R}^p$ corresponde a las p salidas y $u_k \in \mathbb{R}^m$ a las m entradas controlables. Además, $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$ y $C \in \mathbb{R}^{p\times n}$ representan las dinámicas del sistema, mientras que D es cero dado que, para este caso en particular, la salida es independiente de la entrada. La matriz A describe la influencia del estado actual en el estado siguiente, B describe la influencia de la entrada actual en el estado siguiente, y C describe la relación entre las salidas y los estados del sistema. Inicialmente se asume que todos los estados del sistema son medibles para el instante k.

El control de un sistema discreto modelado por (2.5) debe tener la capacidad de llevar la salida del sistema (y_k) a un valor de referencia definido por diseño (r_k). Dicho esto, se define

$$e_k = r_k - y_k, \tag{2.6}$$

con $e_k \in \mathbb{R}^n$ como error de seguimiento de referencia. Con ello se espera que en estado estacionario $e_k = 0$, o bien, $y_k = r_k$.

Si se considera una referencia en el origen (*i.e.*, $r_k = 0$), entonces $e_k = -y_k$. Luego, para penalizar las desviaciones del punto de equilibrio se define una función de costo cuadrático

$$J_k = y_k^{\mathrm{T}} y_k. \tag{2.7}$$

Al sustituir la expresión de y_k de 2.5 en 2.7, se obtiene

$$J_k = x_k^{\mathrm{T}} \underbrace{C^{\mathrm{T}} C}_{\Omega} x_k = x_k^{\mathrm{T}} \Omega x_k, \tag{2.8}$$

donde $\Omega \in \mathbb{R}^{n\times n}$ contiene los pesos cuadráticos asociados a los estados con respecto a la salida. Adicionalmente, existen limitaciones definidas por los componentes utilizados, lo que restringe la actuación sobre el sistema considerando sobretensión, sobrecorriente, pérdidas, entre otros. Es por esto que se diseña un funcional de costos que incluya las entradas de control para evitar dinámicas indeseadas

$$J_k = x_k^{\mathrm{T}}\Omega x_k + u_k^{\mathrm{T}}\Gamma u_k, \tag{2.9}$$

donde la matriz $\Gamma \in \mathbb{R}^{m\times m}$ contiene los costos asociados a la actuación. Esta matriz se obtiene a través de un proceso de diseño previo a la simulación y deben ser escalados para que se encuentren en un orden de magnitud similar a los costos de los estados.

Dado que CCS-MPC utiliza un horizonte de predicción para estimar los estados futuros se generaliza el funcional de costos con un horizonte de predicción igual a N según

$$J(x_k, u_k) = \sum_{k=0}^{N-1} \left(x_k^T \Omega x_k + u_k^T \Gamma u_k \right) + x_N^T \Omega_N x_N. \tag{2.10}$$

2.5.2. Modelo de optimización

El modelo de optimización utilizado en la formulación de CCS-MPC propuesta, se basa en el uso de un modelo convexo tipo QP con restricciones de desigualdad el cual se define de la siguiente forma

$$\begin{aligned} \min_{\mathbf{x}} \quad & \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + \mathbf{q}^{\mathrm{T}}\mathbf{x} \\ \text{sujeto a} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{c}. \end{aligned} \tag{2.11}$$

donde $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{Q} \in \mathbb{R}^{n\times n}$, $\mathbf{q} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{n\times n}$ y $\mathbf{c} \in \mathbb{R}^n$. En particular, el funcional de costos de (2.11) es equivalente a (2.10), donde $\mathbf{x} = \begin{bmatrix} x_k & x_N & u_k \end{bmatrix}^{\mathrm{T}}$ y $\mathbf{Q} = \operatorname{diag}\left(\begin{bmatrix} \Omega & \Omega_N & \Gamma \end{bmatrix}\right)$. Cabe destacar que, para que el problema sea convexo, la matriz hessiana de $\mathbf{Q}$ debe ser semidefinida positiva.

La solución que ofrece MATLAB para resolver problemas QP es una función nativa llamada *quadprog*. Esta función, permite resolver problemas que contengan restricciones lineales, y dado que nuestra formulación considera restricciones de este tipo, lo convierte el candidato ideal para utilizarlo como *solver* de referencia. Dicho esto, se diseñan distintas configuraciones utilizando ADMM en una etapa de simulación para medir el desempeño con el *solver* de referencia, proceso que requiere la comprensión del desarrollo teórico del algoritmo ADMM.

2.6. *Alternating Direction Method of Multipliers* (ADMM)

El algoritmo ADMM toma la forma de un procedimiento de descomposición-coordinación, donde se coordinan subproblemas pequeños de manera local para encontrar una solución a un problema más grande de manera global [27]. En este contexto, ADMM puede ser visto como un intento de combinar las ventajas de los métodos de descomposición dual y Lagrangiano aumentado para problemas con restricciones.

2.6.1. Reformulación para Implementación de ADMM

La utilización de ADMM requiere de un modelo de optimización con restricciones de igualdad [27]. Dicho esto, se debe presentar una reformulación al problema QP con restricciones

de desigualdad presentado en (2.11). Es por esto que se crea una variable de holgura definida como $\mathbf{z}$, la cual presenta el mismo tamaño que $\mathbf{x}$ [18], obteniendo un problema equivalente

$$\begin{aligned} \min_{\mathbf{x}\,\mathbf{z}} \quad & \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + \mathbf{q}^{\mathrm{T}}\mathbf{x} + \mathrm{I}_{\mathbb{Z}}(\mathbf{z}) \\ \text{sujeto a:} \quad & \mathbf{A}\mathbf{x} + \mathrm{I}\mathbf{z} = \mathbf{c} \\ & \mathbb{Z} = \{\mathbf{z} : \mathbf{z} \geq 0\}, \end{aligned} \tag{2.12}$$

donde $\mathrm{I}_{\mathbb{Z}}(\mathbf{z})$ es la función indicatriz de $\mathbb{Z}$, es decir,

$$\mathrm{I}_{\mathbb{Z}}(\mathbf{z}) = \begin{cases} 0 & \text{si } \mathbf{z} \in \mathbb{Z} \\ \infty & \text{si } \mathbf{z} \notin \mathbb{Z}. \end{cases}$$

Basándose en la reformulación presentada en (2.12) se utiliza el método de Lagrangiano Aumentado [28, 29],

$$L_\rho(\mathbf{x}, \mathbf{z}, \mathbf{u}) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + \mathbf{q}^{\mathrm{T}}\mathbf{x} + \mathrm{I}_{\mathbb{Z}}(\mathbf{z}) + \frac{\rho}{2}\|\mathbf{A}\mathbf{x} + \mathrm{I}\mathbf{z} - \mathbf{c} + \mathbf{u}\|_2^2,$$

donde $\rho > 0$ es un parámetro y $\mathbf{u}$ es la variable dual escalada ($\mathbf{y}$ como variable dual, $\mathbf{u} = \mathbf{y}/\rho$).

2.6.2. Selección de ρ

La selección de ρ se obtiene mediante el análisis realizado en [18] de *Step-size selection with respect to 2-norm* el cual se calcula utilizando las matrices $\mathbf{Q}$ y $\mathbf{A}$ del modelo planteado en (2.12). Básicamente, se define la descomposición singular de $\mathbf{A}$ y luego de un trabajo algebraico se obtienen las raíces de un polinomio de grado cuatro, donde se utiliza la primera raíz positiva. Una de las ventajas de este método, en conjunto con las características del modelo de optimización, es que las matrices $\mathbf{Q}$ y $\mathbf{A}$ son constantes, por lo que se calcula ρ una vez de manera offline y se almacena el resultado.

2.6.3. Pasos iterativos del solver

Para resolver el sistema se requiere de un proceso iterativo en cada una de las variables de la función $L_\rho(\mathbf{x}, \mathbf{z}, \mathbf{u})$. Estos pasos se detallan a continuación.

Minimización de x

Primero, para minimizar en torno a $\mathbf{x}$, se deriva la función $L_\rho(\mathbf{x}, \mathbf{z}, \mathbf{u})$ en función de $\mathbf{x}$ y, dado que se busca un punto de inflexión, esta se iguala a cero. Debido a que el sistema es convexo dadas las condiciones previamente mencionadas, al encontrar un punto de inflexión, este punto representa un mínimo de la función. De esta forma, se obtiene la siguiente ecuación

$$\mathbf{v} = \mathbf{z} - \mathbf{c} + \mathbf{u} \tag{2.13}$$

$$\frac{dL_\rho}{d\mathbf{x}} = \mathbf{Q}\mathbf{x} + \mathbf{q}^{\mathrm{T}} + \rho\mathbf{A}^T(\mathbf{A}\mathbf{x} + \mathbf{v}) = 0$$

$$\mathbf{x} = (\mathbf{Q} + \rho\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}(-\rho\mathbf{A}^{\mathrm{T}}\mathbf{v} - \mathbf{q}^{\mathrm{T}}). \tag{2.14}$$

Dado que las matrices $\mathbf{Q}$ y $\mathbf{A}$ son fijas en el modelo de optimización, se tiene que

$$R_{\text{inv}} = (\mathbf{Q} + \rho\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1},$$

se puede precalcular y almacenar su resultado en memoria. Por consiguiente, utilizando el valor prealmacenado de R_{inv} y el uso de la variable auxiliar $\mathbf{v}$ presentado en (2.13), el paso de minimización en $\mathbf{x}$ queda definido según

$$\mathbf{x} = R_{\text{inv}}(-\rho\mathbf{A}^{\text{T}}\mathbf{v} - \mathbf{q}^{\text{T}}). \tag{2.15}$$

De esta manera, se logra un paso iterativo que sólo requiere multiplicación y adición entre matrices y vectores.

Proyección euclidiana de z

El segundo paso del proceso iterativo consiste en evaluar $\pi_{\mathbb{Z}}(\mathbf{z})$, es decir, la proyección euclidiana de $\mathbf{z}$ en $\mathbb{Z}$. Este paso es computacionalmente ligero, ya que si el conjunto $\mathbb{Z}$ se encuentra acotado con restricciones tipo caja, se limitan los valores de $\mathbf{z}$ para que se encuentre dentro de la región factible, esto es,

$$\begin{gathered}\mathbb{Z} \in [\alpha, \beta] \\ \pi_{\mathbb{Z}}(\mathbf{z}) = \text{máx}(\alpha, \text{mín}(\mathbf{z}, \beta)).\end{gathered}$$

Ya que $\mathbf{z}$ describe una variable de holgura que debe ser semidefinida positiva, equivalente a $\mathbb{Z} \in [0, \infty)$, se tiene que

$$\begin{aligned}\mathbf{z} &= \pi_{\mathbb{Z}}(-\mathbf{A}\mathbf{x} - \mathbf{u} + \mathbf{c}) \\ \mathbf{z} &= \text{máx}(0, -\mathbf{A}\mathbf{x} - \mathbf{u} + \mathbf{c})\end{aligned} \tag{2.16}$$

Actualización de u

El último paso iterativo consiste en actualizar la variable dual escalada según

$$\mathbf{u} = \mathbf{u} + (\mathbf{A}\mathbf{x} + \mathbf{z} - \mathbf{c}) \tag{2.17}$$

Pseudocódigo

Los pasos previamente mencionados se resumen a través del pseudocódigo presentado en Algoritmo 1. En este algoritmo, se puede verificar que la inversión de matriz se encuentra fuera del proceso iterativo. Además, el proceso iterativo consta de álgebra matriz-vector basado en multiplicaciones y adiciones. La cantidad de iteraciones que se desean utilizar se define en un parámetro N_{ADMM}, el cual se debe escoger según el desempeño del algoritmo en la aplicación deseada. Una de las principales ventajas de este algoritmo de optimización para una implementación en FPGA es que la complejidad matemática permite la implementación a través de arquitecturas altamente paralelizables y especializadas.

Algoritmo 1 ADMM para reformulación de problema QP con restricciones de desigualdad

Input: $\mathbf{Q}, \mathbf{q}, \mathbf{A}, \mathbf{c}, N_{\text{ADMM}}, \mathbf{x_0}, \mathbf{z_0}, \mathbf{u_0}, \rho_0$
Output: $\mathbf{x}$
  Initialize:
    $\mathbf{x} \leftarrow \mathbf{x}_0$
    $\mathbf{z} \leftarrow \mathbf{z}_0$
    $\mathbf{u} \leftarrow \mathbf{u}_0$
    $\rho \leftarrow \rho_0$
  $\mathbf{R}_{\text{inv}} = (\mathbf{Q} + \rho \mathbf{A}^{\text{T}} \mathbf{A})^{-1}$
  for $k = 0$ **to** N_{ADMM} **do**
    $\mathbf{v}^k = \mathbf{z}^k - \mathbf{c} + \mathbf{u}^k$ ▷ (2.13)
    $\mathbf{x}^{k+1} = \mathbf{R}_{\text{inv}}(-\rho \mathbf{A}^{\text{T}} \mathbf{v}^k - \mathbf{q}^{\text{T}})$ ▷ (2.14)
    $\mathbf{z}^{k+1} = \text{máx}(0, -\mathbf{A}\mathbf{x}^{k+1} - \mathbf{u}^k + \mathbf{c})$ ▷ (2.16)
    $\mathbf{u}^{k+1} = \mathbf{u}^k + (\mathbf{A}\mathbf{x}^{k+1} + \mathbf{z}^{k+1} - \mathbf{c})$ ▷ (2.17)
  end for
  return $\mathbf{x}$

Una vez definida la aplicación objetivo y los componentes del esquema de control, que es donde nos enfocaremos en este trabajo, se requiere de una etapa de simulación para analizar el desempeño del *solver* propuesto comparándolo con un *solver* de referencia de MATLAB para resolver problemas QP, como *quadprog*. De esta forma, se puede dar paso a una etapa de desarrollo de código para una posterior implementación, considerando tanto el desempeño como el comportamiento dinámico del sistema ante los distintos casos propuestos.

3 Simulación de sistema de referencia

A través de la simulación del sistema de referencia se puede obtener información sobre la dinámica del sistema y el desempeño del lazo de control, información con la que se sustentan las decisiones de diseño para implementar el controlador. Bajo este contexto, se desarrolla una simulación donde se varía la cantidad el número de iteraciones de ADMM con la finalidad de analizar la respuesta dinámica del sistema y así definir una base para la implementación en FPGA. En este capítulo se presenta una simulación en MATLAB/Simulink de una variación del sistema desarrollado por el estudiante del programa Doctorado en Ingeniería Electrónica Reinier Lopez en [12], donde se reemplazan los *solvers* de prueba por una implementación de ADMM basada en la información extraída de [27] y [18].

Las simulaciones de este capítulo se encuentran en la carpeta matlab[1] del repositorio.

3.1. Preparación para simulación

3.1.1. Parámetros de simulación

El sistema de potencia presentado en la Figura 2.1a se simula con los parámetros mostrados en la Tabla 3.1, los cuales se basan en el trabajo presentado en [12].

3.1.2. Proceso para crear modelo espacio estado discreto

Para la creación de un modelo espacio estado se comienza con la definición de un modelo espacio estado en variable continua

$$\begin{aligned} \frac{dx}{dt} &= Ax + Bu + B_p d \\ y &= Cx, \end{aligned} \tag{3.1}$$

donde $x \in \mathbb{R}^n$ y $\frac{dx}{dt} \in \mathbb{R}^n$ representan los n estados y sus derivadas respectivamente, $y \in \mathbb{R}^p$ corresponde a las p salidas, $u \in \mathbb{R}^m$ a las m entradas controlables y $d \in \mathbb{R}^m$ a las m perturbaciones. Además, $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $B_p \in \mathbb{R}^{n\times m}$ y $C \in \mathbb{R}^{p\times n}$, las cuales tienen una

[1] `https://gitlab.com/jdjotad/ipd-500-juan-escarate/-/tree/main/matlab`

Tabla 3.1: Parámetros de Simulación

Parámetros del sistema	**Valor**
Bus DC v_{DC}	100 V
Voltaje referencia coordenada d V_{ref_d}	50 V
Voltaje referencia coordenada q V_{ref_q}	0 V
Corriente nominal I_n	8 A
Inductancia del filtro L_f	3 mH
Capacitor del filtro C_f	15 μF
Resistencia del filtro R_f	65 $m\Omega$
Frecuencia nominal f	50 Hz
Tiempo de muestreo Δ	200 μs
Resistencia de carga (t $<$ 0.2s) R_L	23.6 Ω
Resistencia de carga (t $\geq$ 0.2s) R_L	4.72 Ω
Parámetros del diseño de MPC	**Valor**
Horizonte de predicción N	2
Matriz de peso de la entrada Γ	$100\mathbb{I}_2$
Matriz de peso de los estados Ω	$\mathrm{diag}\Big(\begin{bmatrix}100 & 100 & 1 & 1\end{bmatrix}\Big)$

representación numérica según se detalla a continuación

$$A = \begin{bmatrix} -R_f/L_f & 2\pi f & -1/L_f & 0 \\ -2\pi f & -R_f/L_f & 0 & -1/L_f \\ 1/C_f & 0 & 0 & 2\pi f \\ 0 & 1/C_f & -2\pi f & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 1/L_f & 0 \\ 0 & 1/L_f \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$B_p = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ -1/C_f & 0 \\ 0 & -1/C_f \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

y los estados, actuación y perturbación del sistema se definen según

$$x = \begin{bmatrix} I_{fd} \\ I_{fq} \\ V_{cd} \\ V_{cq} \end{bmatrix}$$
$$u = \begin{bmatrix} V_{md} \\ V_{mq} \end{bmatrix}$$
$$d = \begin{bmatrix} I_{od} \\ I_{oq} \end{bmatrix}.$$

Luego, se define un modelo espacio estado de variable discreta según

$$\begin{aligned} x_{k+1} &= A_d x_k + B_d u_k + B_{pd} d_k \\ y_k &= C x_k, \end{aligned} \tag{3.2}$$

donde la obtención de A_d, B_d y B_{pd} proviene del uso del comando `c2d()` de MATLAB, el cual convierte un modelo de tiempo continuo a discreto,

```
[Ad, Bd] = c2d(A,B,T);
[~, Bpd] = c2d(A,Bp,T);
```

con $T = \Delta = 200\text{e}{-6}$.

Mientras que Γ y Ω se obtienen mediante sintonización, Ω_N proviene del uso del comando *dlqr*() de MATLAB, el cual se define como un regulador lineal cuadrático de feedback de estados para un sistema de espacio de estados de tiempo discreto,

```
[~,Omega_N,~] = dlqr(Ad,Bd,Omega,Gamma);
```

Con la obtención de las matrices que conforman el modelo espacio estado discreto, se crea la matriz **Q** asociada a los pesos de los costos quadráticos de los modelos (2.11) y (2.12). De esta forma, se agrupa Γ, Ω y Ω_N de la siguiente forma:

$$Q_{16\text{x}16} = \begin{bmatrix} \Omega_{4\text{x}4} & 0 & \dots & \dots & 0 \\ 0 & \Omega_{4\text{x}4} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \Omega_{N\,4\text{x}4} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \Gamma_{2\text{x}2} & 0 \\ 0 & \dots & \dots & 0 & \Gamma_{2\text{x}2} \end{bmatrix},$$

obteniendo una matriz que tiene valores únicamente en su diagonal y es de tamaño 16x16.

3.1.3. Restricciones de voltaje y corriente

Las restricciones de voltaje de actuación se definen como un hexágono en coordenadas $\alpha\beta$ [12] según

$$\begin{bmatrix} -\frac{2}{\sqrt{3}}V_{dc} \\ -\frac{2}{\sqrt{3}}V_{dc} \\ -\frac{1}{\sqrt{3}}V_{dc} \end{bmatrix} \leq \begin{bmatrix} -\frac{3}{\sqrt{3}}V_{dc} & 1 \\ \frac{3}{\sqrt{3}}V_{dc} & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} V_{m\alpha} \\ V_{m\beta} \end{bmatrix} \leq \begin{bmatrix} \frac{2}{\sqrt{3}}V_{dc} \\ \frac{2}{\sqrt{3}}V_{dc} \\ \frac{1}{\sqrt{3}}V_{dc} \end{bmatrix}.$$

Estas restricciones pueden ser reescritas en coordenadas dq (ver sección Transformada dq) de la siguiente forma

$$\begin{bmatrix} -\frac{2}{\sqrt{3}}V_{dc} \\ -\frac{2}{\sqrt{3}}V_{dc} \\ -\frac{1}{\sqrt{3}}V_{dc} \end{bmatrix} \leq \begin{bmatrix} -\frac{3}{\sqrt{3}}V_{dc} & 1 \\ \frac{3}{\sqrt{3}}V_{dc} & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta(t)) & -\sin(\theta(t)) \\ \sin(\theta(t)) & \cos(\theta(t)) \end{bmatrix} \begin{bmatrix} V_{md} \\ V_{mq} \end{bmatrix} \leq \begin{bmatrix} \frac{2}{\sqrt{3}}V_{dc} \\ \frac{2}{\sqrt{3}}V_{dc} \\ \frac{1}{\sqrt{3}}V_{dc} \end{bmatrix}.$$

Sin embargo, se diseña una simplificación de las restricciones de voltaje con el fin de que las matrices del modelo de optimización sean en su mayoría fijas, pudiendo ser precalculadas y almacenadas en memoria. A pesar de la pérdida de precisión en la actuación del sistema, se logra seguir la referencia de voltaje en el lazo de control. Aproximando las restricciones de voltaje un decágono regular, de la misma forma que [12] aproxima las restricciones de corriente. De esta forma se tiene la siguiente restricción para la actuación de voltaje

$$\underbrace{\frac{V_{dc}}{\sqrt{3}}\begin{bmatrix} -3.078 \\ -3.078 \\ -(\sin(\pi/5)+0.726\cos(\pi/5)) \\ -(\sin(\pi/5)+0.726\cos(\pi/5)) \\ -\sin(2\pi/5) \end{bmatrix}}_{u_{lb}} \leq \underbrace{\begin{bmatrix} 3.078 & 1 \\ -3.078 & 1 \\ 0.726 & 1 \\ -0.726 & 1 \\ 0 & 1 \end{bmatrix}}_{H_v} \underbrace{\begin{bmatrix} V_{md} \\ V_{mq} \end{bmatrix}}_{u} \leq \underbrace{\frac{V_{dc}}{\sqrt{3}}\begin{bmatrix} 3.078 \\ 3.078 \\ (\sin(\pi/5)+0.726\cos(\pi/5)) \\ (\sin(\pi/5)+0.726\cos(\pi/5)) \\ \sin(2\pi/5) \end{bmatrix}}_{u_{ub}}$$

$$u_{lb} \leq H_v u \leq u_{ub}.$$

Las restricciones de corriente se representan por una circunferencia de radio $I_n = 8\ A$ en coordenadas dq. En [12] se presenta una aproximación por un decágono regular, lo que simplifica las restricciones de corriente a un área delimitada por diez rectas, las cuales pueden ser representadas mediante las siguientes desigualdades en coordenadas dq

$$\begin{aligned} -3.078I_n \leq\ & 3.078I_{fd} + I_{fq} \leq 3.078I_n \\ -3.078I_n \leq\ & -3.078I_{fd} + I_{fq} \leq 3.078I_n \\ -I_n(\sin(\pi/5)+0.726\cos(\pi/5)) \leq\ & 0.726I_{fd} + I_{fq} \leq I_n(\sin(\pi/5)+0.726\cos(\pi/5)) \\ -I_n(\sin(\pi/5)+0.726\cos(\pi/5)) \leq\ & -0.726I_{fd} + I_{fq} \leq I_n(\sin(\pi/5)+0.726\cos(\pi/5)) \\ -I_n\sin(2\pi/5) \leq\ & I_{fq} \leq I_n\sin(2\pi/5). \end{aligned}$$

Dado que el vector de estados se define como $x = \begin{bmatrix} I_{fd} & I_{fq} & V_{cd} & V_{cq} \end{bmatrix}^{\mathrm{T}}$, se reescriben las restricciones de corriente según

$$\underbrace{\begin{bmatrix} -3.078I_n \\ -3.078I_n \\ -I_n(\sin(\pi/5)+0.726\cos(\pi/5)) \\ -I_n(\sin(\pi/5)+0.726\cos(\pi/5)) \\ -I_n\sin(2\pi/5) \end{bmatrix}}_{x_{lb}} \leq \underbrace{\begin{bmatrix} 3.078 & 1 \\ -3.078 & 1 \\ 0.726 & 1 \\ -0.726 & 1 \\ 0 & 1 \end{bmatrix}}_{H_i} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}}_{M_i} \underbrace{\begin{bmatrix} I_{fd} \\ I_{fq} \\ V_{cd} \\ V_{cq} \end{bmatrix}}_{x} \leq \underbrace{\begin{bmatrix} 3.078I_n \\ 3.078I_n \\ I_n(\sin(\pi/5)+0.726\cos(\pi/5)) \\ I_n(\sin(\pi/5)+0.726\cos(\pi/5)) \\ I_n\sin(2\pi/5) \end{bmatrix}}_{x_{ub}}$$

$$x_{lb} \leq H_i M_i x \leq x_{ub}.$$

3.1.4. Cálculo de estados y actuación en estado estacionario

El desarrollo planteado en esta sección se basa en la teoría presentada por Rawlings et al. [30]. Comúnmente, se plantea un objetivo de control con retroalimentación que lleve las salidas

medibles de un sistema a un valor fijo, problema conocido como "seguimiento de referencia". Para ello, se utiliza como base un modelo espacio-estado como el presentado en (3.2).

Primero se tiene el modelo espacio-estado en estado estacionario[2], sin considerar la salida,

$$\begin{aligned} x_{s+1} &= x_s \\ x_s &= A_d x_s + B_d u_s + B_{pd} d_s, \end{aligned}$$

el cual se puede reescribir como

$$\begin{bmatrix} \mathrm{I} - A & -B \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} B_{pd} d_s \end{bmatrix}. \tag{3.3}$$

Luego, se impone el requerimiento de que el estado estacionario satisfaga

$$H_x C x_s = y_{\mathrm{ref}},$$

donde H_x se utiliza para seleccionar las salidas medibles que participan en el diseño del lazo de control, Cx_s representa la salida en estado estacionario (y_s) y y_{ref} es el valor de referencia. De esta forma, quedaría un modelo espacio-estado de la siguiente forma

$$\begin{aligned} x_{s+1} &= x_s \\ x_s &= A_d x_s + B_d u_s + B_{pd} d_s \\ H_x C x_s &= y_{\mathrm{ref}}. \end{aligned}$$

Este sistema de ecuaciones puede ser reescrito en forma matricial según

$$\begin{bmatrix} \mathrm{I} - A & -B \\ H_x C & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} B_{pd} d_s \\ y_{\mathrm{ref}} \end{bmatrix}. \tag{3.4}$$

Con el sistema de ecuaciones presentado en (3.4), se obtiene el valor de los estados y las actuaciones en estado estacionario de la siguiente forma

$$\begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} \mathrm{I} - A & -B \\ H_x C & 0 \end{bmatrix}^{-1} \begin{bmatrix} B_{pd} d_s \\ y_{ref} \end{bmatrix}. \tag{3.5}$$

La función de MATLAB que replica el algoritmo previamente desarrollado se muestra en el Código 3.1.

Finalmente, se realiza un manejo de matrices para definir las entradas al solver de optimización las cuales se detallan en el código *CCS-MPC.m* el cual se encuentra en la carpeta matlab del repositorio.

Posterior a la definición de las matrices que componen las entradas al solver de optimización, se desarrolla un esquema del sistema utilizando los parámetros definidos en la Tabla 3.1 para así medir el desempeño del lazo de control cuando se reemplaza el solver *quadprog* por ADMM.

[2]Las variables en estado estacionario contienen un subíndice s, por ejemplo x_s, u_s y d_s para los estados, las actuaciones y perturbaciones en estado estacionario respectivamente.

Código 3.1: Función de Matlab para calcular los valores de los estados y la actuación en estado estacionario.

```
% Ad -> Matriz Ad de modelo espacio estado discreto
% Bd -> Matriz Bd de modelo espacio estado discreto
% Bpd -> Matriz Bpd de modelo espacio estado discreto
% Cd -> Matriz Cd de modelo espacio estado discreto
% per -> Perturbacion
% yref -> Referencia de salida
function [xs, us] = Infinity(Ad, Bd, Bpd, Cd, per, yref)
    H_k     = [ 0 0 1 0;  % Seleccionar vd y vq para hacer seguimiento
                0 0 0 1]; % de referencia
    % Asolve * infy =  Csolve
    Asolve  = [eye(length(Ad))-Ad          -Bd;
                            H_k*Cd   zeros(2,2)];
    Csolve  = [Bpd*per;
                  yref];

    % infy = Asolve \ Csolve
    infy  = linsolve(Asolve,Csolve);
    % infy = [xs; us]
    xs = infy(1:size(Ad,1));
    us = infy(size(Ad,1)+1:end);
end
```

3.2. Simulación del sistema de referencia

La simulación implementada en Simulink mantiene las conexiones mostradas en el sistema de referencia (revisar diagrama de Simulink y Figura 2.1a). Con las señales obtenidas de la simulación, se realiza un análisis del desempeño del lazo de control al utilizar ADMM, variando las iteraciones del algoritmo, para comparar su respuesta dinámica con la obtenida al utilizar el *solver quadprog* de MATLAB.

3.2.1. Desempeño del lazo con *quadprog*

En la Figura 3.1 se muestra el desempeño del lazo de control utilizando el *solver quadprog*. En esta figura se presenta el seguimiento de voltaje en coordenadas *dq* considerando que el voltaje de referencia es

$$V_{ref_{dq}} = 50 + 0j$$
$$V_{ref_d} = 50 \text{ V}$$
$$V_{ref_q} = 0 \text{ V}.$$

Como se puede visualizar, en el gráfico de voltaje existe un seguimiento de referencia en el tramo temporal t $<$ 0.2 s. Al llegar a estado estacionario en t $<$ 0.2 s, se cumple lo siguiente

$$V_{cd} \approx 50 \text{ V}$$
$$|I_{f_{dq}}| = \frac{V_{cd}}{R_L} \approx \frac{50}{23.6} = 2.1186 \text{ A}$$

Luego de realizar un cambio de carga de 23.6 Ω a 4.72 Ω en 0.2 s, existe una saturación en la magnitud de corriente, ya que

$$|I_{f_{dq}}| = \frac{V_{cd}}{R_L} \approx \frac{50}{4.72} = 10.5932 \text{ A} > I_n = 8 \text{ A}.$$

Debido a que el modelo de optimización considera la saturación de la corriente al valor nominal como una restricción, se tiene que la corriente no debería superar los 8 A. De esta forma, al tener una corriente saturada al valor nominal se obtiene un voltaje V_{cd} según

$$|I_{f_{dq}}| \approx 8$$
$$V_{cd} = |I_{f_{dq}}|R_L \approx 8 \cdot 4.72 = 37.76 \text{ V}$$

El comportamiento dinámico del voltaje de salida se puede ver en la Figura 3.2 con v_{c_a}, v_{c_b} y v_{c_c}, representando el voltaje del condensador de salida las fases a, b y c según la Figura 2.1a. Adicionalmente, al realizar un zoom en el cambio de carga para $t = 0.2$ s, se visualiza un transiente antes de llegar a estado estacionario, donde se presenta una disminución en la magnitud de la señal, con respecto al valor de referencia de 50 V. Esto se conoce como error de estado estacionario, el cual se puede ver tanto en el seguimiento de referencia en la Figura 3.1 como en la respuesta dinámica en la Figura 3.2, ocasionado por la saturación de corriente.

Al utilizar *quadprog*, no existe un sobrepaso en la corriente, consecuencia de la correcta limitación de corriente, y por consiguiente, la solución del problema de optimización se encuentra dentro de la región factible.

La simulación con el solver *quadprog* se utiliza como referencia para medir el desempeño al variar la cantidad de iteraciones de ADMM.

3.2.2. Desempeño del lazo con ADMM

La implementación del algoritmo ADMM se basa en el Algoritmo 1, el cual utiliza una cantidad de iteraciones fijas y el parámetro ρ es precalculado de manera offline. Por otra parte, en nuestro sistema, las matrices y vectores del modelo 2.12, a excepción de $\mathbf{c}$, son constantes. Por lo que, se almacenan en memoria para sólo computar el vector $\mathbf{c}$ en cada intervalo de muestreo. Con esta información, se diseñan pruebas con un máximo de 50 iteraciones para ADMM apuntando a evaluar el *tradeoff* entre número de iteraciones y desempeño de control para la aplicación.

Se puede analizar de manera visual el desempeño del seguimiento de voltaje en la Figura 3.3, mientras que el comportamiento de la corriente se muestra en la Figura 3.4. En particular, la velocidad de convergencia del voltaje varía según la cantidad de iteraciones de ADMM, mientras mayor las iteraciones, más rápida es la convergencia. Además, la respuesta dinámica de corriente muestra un sobrepaso a la corriente nominal, el cual disminuye en magnitud y tiempo a medida que se aumenta la cantidad de iteraciones definidas para el *solver*.

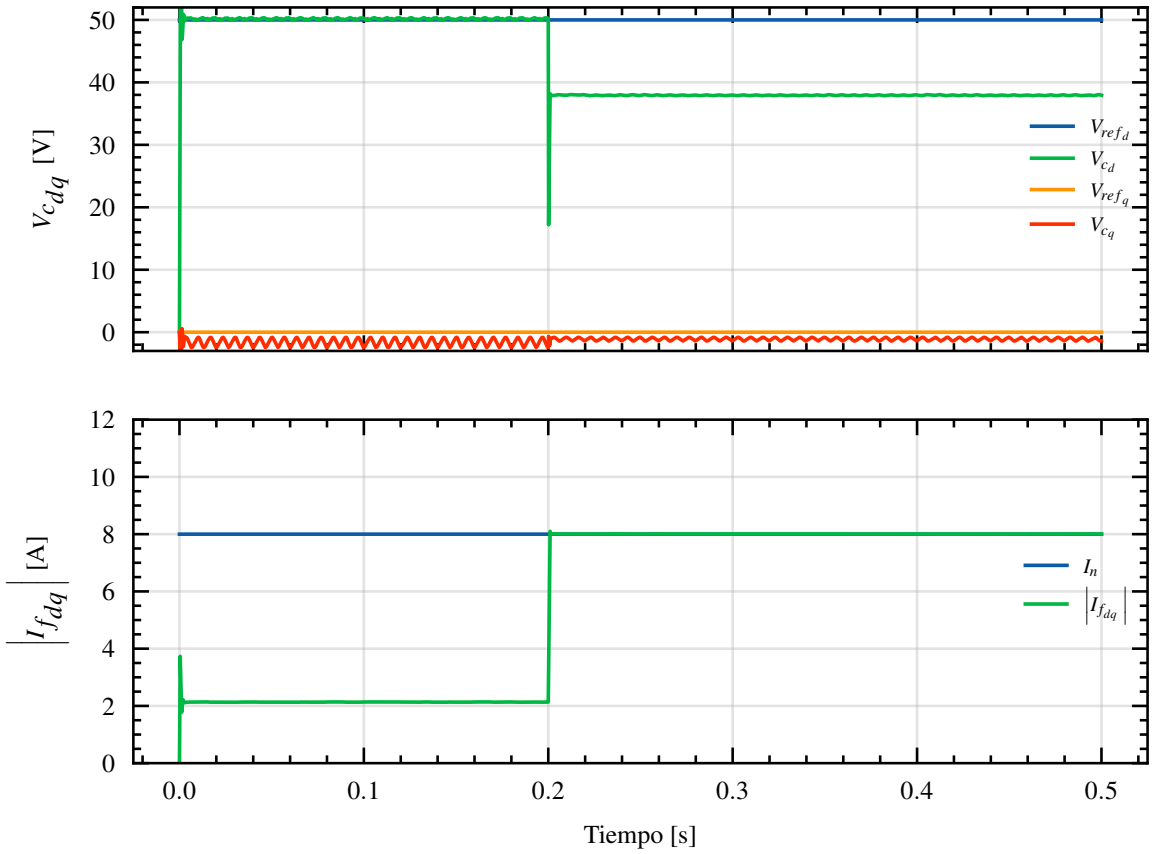


Figura 3.1: Seguimiento de voltaje a referencia (gráfico superior) y comportamiento de corriente según valores nominales (gráfico inferior) utilizando solver *quadprog*.

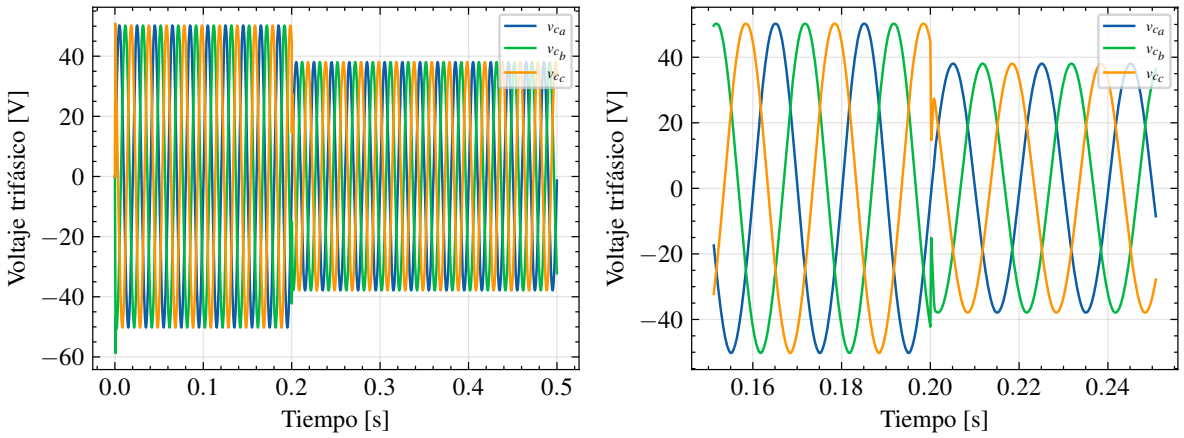


Figura 3.2: (a) Voltaje trifásico de salida utilizando *quadprog*. (b) Zoom de voltaje trifásico de salida utilizando *quadprog*.

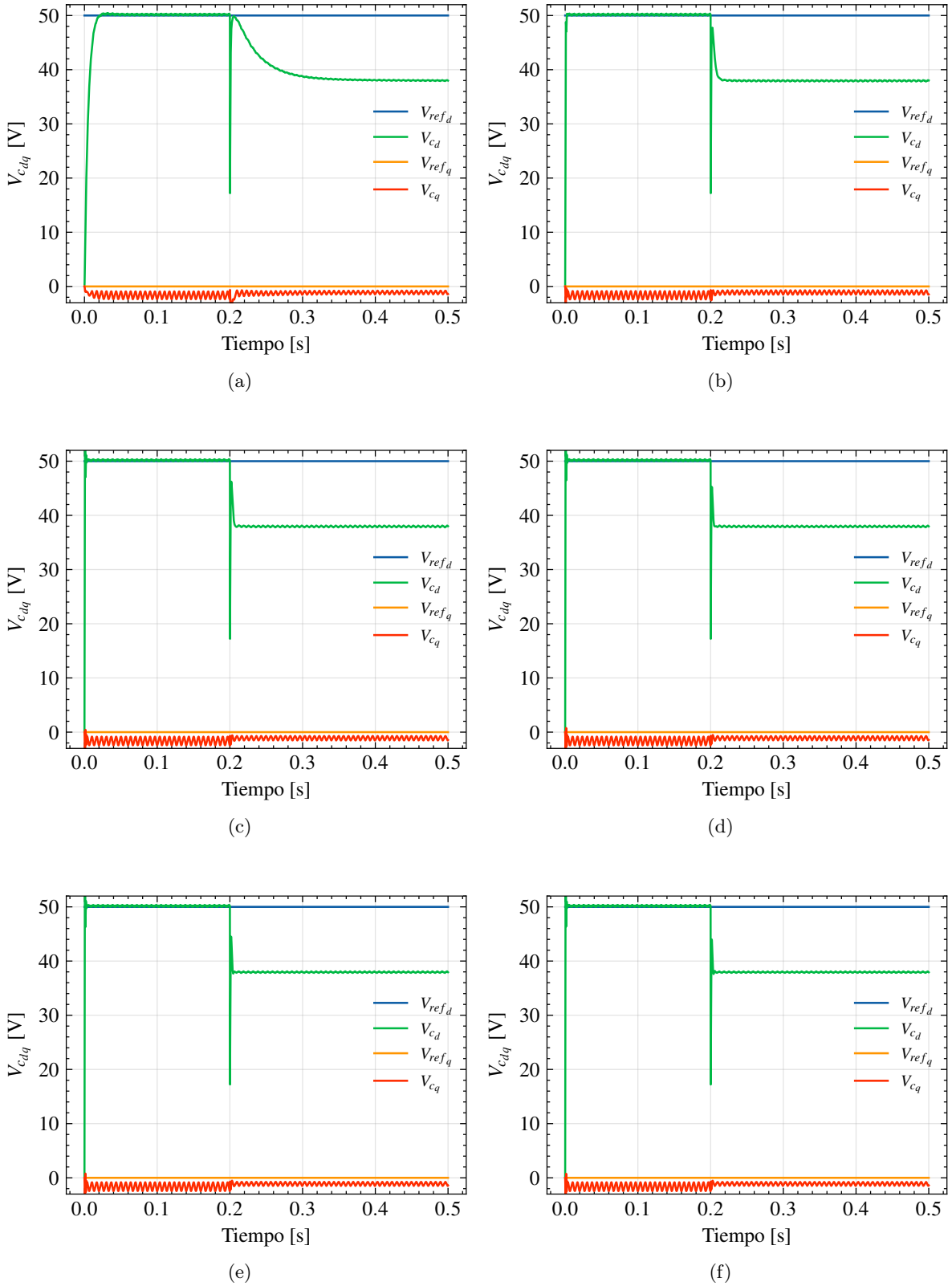


Figura 3.3: Seguimiento de voltaje utilizando solver ADMM con 1 (a), 10 (b), 20 (c), 30 (d), 40 (e) y 50 (f) iteraciones.

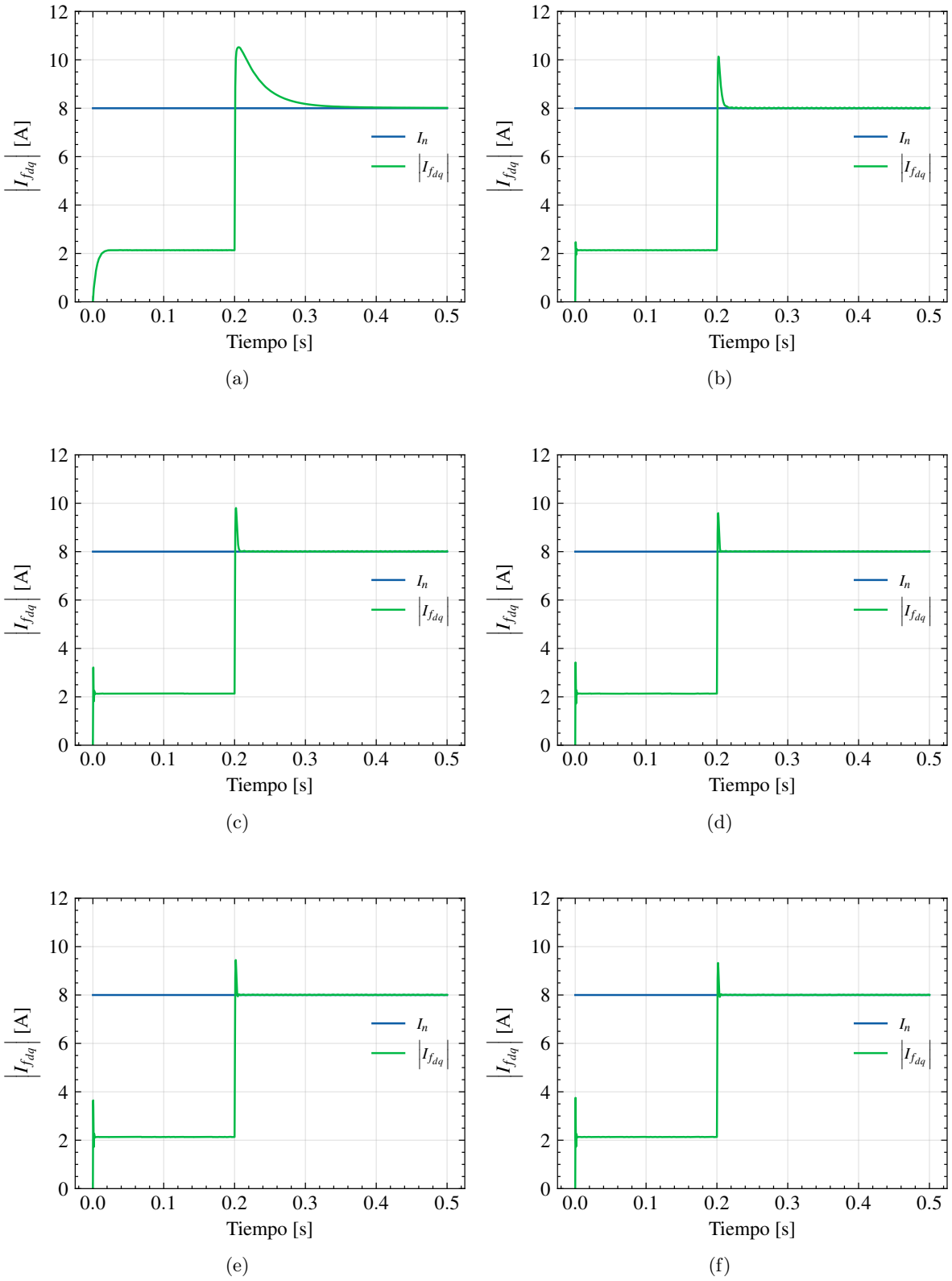


Figura 3.4: Comportamiento dinámico de la corriente del inductor de salida utilizando solver ADMM con 1 (a), 10 (b), 20 (c), 30 (d), 40 (e) y 50 (f) iteraciones.

3.2.3. Métricas de desempeño

Para medir el error al reemplazar *quadprog* por ADMM como *solver* de optimización, se utiliza *Normalized Root Mean Squared Error* (NRMSE), definido como

$$\text{NRMSE} = \frac{\text{ADMM} - \text{quadprog}}{\text{máx}(\text{quadprog}) - \text{mín}(\text{quadprog})}.$$

De esta forma se obtiene un NRMSE para el tramo lineal ($t < 0.2$ s) y saturado ($t \geq 0.2$ s) que se puede ver en la Tabla 3.2, donde el error va disminuyendo a medida que aumentan las iteraciones de ADMM. Sin embargo, hace falta un análisis más detallado para poder verificar las grandes diferencias entre estado transiente y estacionario para cada uno de los dos casos.

Tabla 3.2: NRMSE para comportamiento lineal y saturado.

	Lineal		**Saturado**	
N_{ADMM}	V_{c_d}	V_{c_q}	V_{c_d}	V_{c_q}
1	69.46 %	12.28 %	57.92 %	26.67 %
10	17.03 %	6.16 %	17.57 %	7.94 %
20	8.01 %	3.00 %	11.66 %	7.14 %
30	4.07 %	1.50 %	9.02 %	6.38 %
40	2.09 %	0.93 %	7.49 %	5.60 %
50	1.05 %	0.67 %	6.44 %	4.93 %

Para construir una tabla con el desglose para estado transiente y estacionario, así como poder definir otros parámetros de desempeño utilizados en este trabajo, se debe tener el tiempo en que cada tramo llega a estado estacionario. Para el cálculo del tiempo que demora cada uno de los dos casos en llegar a estado estacionario, se recorre el vector de datos de manera inversa, guardando el instante de tiempo en que el valor de la señal se aleja de una banda del 5 % del valor en estado estacionario (ver Figura 3.5). En la Tabla 3.3 se muestran los tiempos en que el sistema converge a estado estacionario utilizando *quadprog* y las seis configuraciones de ADMM. Con la información obtenida se puede concluir que, al tener un comportamiento lineal, el tiempo que demora en estabilizar la señal no varía considerablemente en la implementación de ADMM, con excepción de la implementación con una iteración. Sin embargo, cuando se realiza el cambio de carga a los 0.2 s, el tiempo que demora en converger es mucho mayor en ADMM que en *quadprog*, variando entre unas 11.5 y 4 veces para 10 y 50 iteraciones de ADMM respectivamente. Además, para 50 iteraciones se demora un total de 3.2 ms en alcanzar el estado estacionario, mientras que con 10 iteraciones se demora 9.2 ms, lo que significa que se obtiene una velocidad de convergencia 2.875 veces mayor con 50 iteraciones.

Por completitud, se incluye el caso de ADMM con 1 iteración, pero no es motivo de análisis dado que las métricas de desempeño y la inspección visual demuestra que su respuesta dinámica no es similar con los otros casos de estudio. Esto último se debe a que el fenómeno de sobrecorriente es indeseado ya sea en magnitud y en tiempo, ya que, en una implementación real, la sobrecorriente puede significar principalmente dos cosas: aumento de pérdidas o, mal funcionamiento de los dispositivos de los dispositivos de electrónica de potencia.

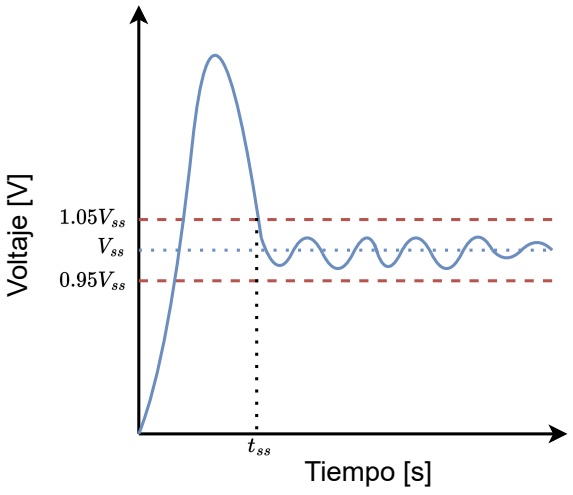


Figura 3.5: Representación gráfica de criterio para definir tiempo que demora en alcanzar estado estacionario.

Tabla 3.3: Tiempo en alcanzar estado estacionario [ms].

	Lineal	**Saturado**
	quadprog	
	1.2	0.8
N_{ADMM}	**ADMM**	
1	12.4	88
10	1.0	9.2
20	1.2	5.4
30	1.6	4.2
40	1.6	3.4
50	1.6	3.2

Para cuantificar la calidad de la señal de voltaje de salida se mide el THD, el cuál presenta una relación entre el contenido armónico de la señal y la primera armónica. El valor de THD se ubica entre 0 % e infinito, siendo 0 % el mejor caso. Para ello se requiere el espectro en frecuencia de la señal en estado estacionario, y con esta información se plantea la ecuación de THD según

$$\text{THD} = \frac{\sum_{i=2}^{50} V_i}{V_1}, \tag{3.6}$$

donde V_1 representa la magnitud en la frecuencia fundamental, y el numerador representa la sumatoria desde la segunda hasta la quincuagésima armónica. En la Tabla 3.5) se muestra que el THD se encuentra por debajo del 1.6 %, indicando que en estado estacionario, la señal resultante presenta un contenido armónico similar en todas las implementaciones. Debido a la similitud en la regulación del voltaje de salida en estado estacionario, el cual se define como el principal objetivo de control, se da paso al análisis de la dinámica del sistema en estado transiente.

El NRMSE para los estados transiente y estacionario se presenta en la Tabla 3.6. Según los resultados asociados al tramo lineal, el error en estado transiente para V_{cd} está por sobre el 100 % hasta las veinte iteraciones de ADMM, disminuyendo hasta un 11.41 % con cincuenta iteraciones. Por otra parte, en el tramo saturado el error en estado transiente comienza en 106.53 % decayendo hasta lograr un 61.96 % con cincuenta iteraciones. Cabe destacar que en estado estacionario, el error en el tramo lineal es siempre menor al tramo con saturación de corriente. Sin embargo, en estado transiente existe el fenómeno de sobrecorriente cuando hay una saturación de corriente, por lo que es esperable que el NRMSE sea mayor al presentar saturación de corriente. Es destacable que, en tramo lineal se obtiene un error del 11.41 % en V_{cd}, mientras que al existir saturación de corriente el valor aumenta 5.43 veces. Esta relación sucede en los casos sobre treinta iteraciones de ADMM, y tiene que ver con la dinámica del sistema antes de alcanzar su estado estacionario y la sobrecorriente al realizar el cambio de carga.

La sobrecorriente (ver Figura 3.4) se puede medir tanto en su magnitud como en el tiempo en que se permanece en ese estado (ver Tabla 3.4). Ya que se incluyen restricciones de corriente en el modelo de optimización, es esperable la ausencia de sobrecorriente. Sin embargo, dado que se implementa el *solver* iterativo ADMM sin criterio de parada, la solución se considera como pseudo óptima. La ventaja de definir una cantidad de iteraciones fija es que, sin importar qué tan cerca o lejos se encuentre de la solución óptima, siempre habrá un tiempo de ejecución determinista. Por otra parte, la principal desventaja es que en caso que se esté lo suficientemente cerca de la solución óptima, no hay forma de detener el proceso iterativo, de manera que existe la posibilidad de que el resultado entregado sea cercano al óptimo y no el óptimo.

Al utilizar *quadprog* como *solver* del problema de optimización, se obtiene la solución óptima para cada intervalo de control. En cambio, la implementación de ADMM debe ser capaz de obtener una solución dentro de un límite temporal, por lo que, como se mencionó previamente, es esperable que las soluciones proporcionadas por este solver sean pseudo óptimas. Es por esto que, en los seis casos de ADMM existe sobrecorriente, la cual presenta un sobrepaso de 1.325 A en el mejor caso, lo cual equivale a un 16.6 % del valor nominal. Adicionalmente, se debe realizar un estudio de la potencia adicional que se consume al utilizar ADMM en vez de *quadprog*.

En un conductor las pérdidas son directamente proporcionales al cuadrado de la corriente que pasa por él. Para este caso, se cuantifican la potencia adicional consumida (pérdidas), provocadas por la sobrecorriente en la respuesta dinámica al utilizar ADMM. Para ello, se analiza el tramo donde existe saturación de corriente para calcular la diferencia de potencia, entre ADMM y quadprog, utilizando el área bajo la curva de corriente al cuadrado. Con ello se definen las siguiente ecuaciones

$$\mathrm{P}_{R_{\mathrm{dif}}} = \frac{1}{t_{ss} - 0.2} \int_{0.2}^{t_{ss}} (|I_{f_{dq}}|_{\mathrm{ADMM}})^2 \; - \; (|I_{f_{dq}}|_{\mathrm{quadprog}})^2 \; dt \left[\frac{\mathrm{W}}{\Omega}\right] \tag{3.7}$$

$$\mathrm{P}_{\mathrm{dif}} = R \cdot \mathrm{P}_{R_{\mathrm{diff}}} [\mathrm{W}] \tag{3.8}$$

donde t_{ss} es el tiempo que demora en alcanzar estado estacionario para cada caso de ADMM (ver Tabla 3.3), $|I_{f_{dq}}|_{\mathrm{ADMM}}$ es la magnitud de la corriente para cada caso de ADMM (ver Figura 3.4), $|I_{f_{dq}}|_{\mathrm{quadprog}}$ es la magnitud de la corriente utilizando el solver *quadprog* (ver Figura 3.1) y R es la resistencia del conductor utilizado. En (3.7) se representa la diferencia de potencia entre ADMM y quadprog en función de la resistencia del conductor. Si se multiplica $\mathrm{P}_{R_{\mathrm{dif}}}$ por la resistencia del conductor se obtiene la diferencia de potencia $\mathrm{P}_{\mathrm{dif}}$ (ver (3.8)).

La Tabla 3.4 contiene información de la diferencia de potencia, proporcional a la resistencia del conductor, según la definición dada en (3.7). Con ello se tiene que existe una constante disminución en la diferencia de potencia que comienza en 19.016 $\left[\frac{\mathrm{W}}{\Omega}\right]$ para una iteración de ADMM y alcanza 10.361 $\left[\frac{\mathrm{W}}{\Omega}\right]$ para cincuenta iteraciones de ADMM.

Para ejemplificar la importancia la diferencia de potencia, se considera un cable de *American Wire Gages* (AWG) igual a 20, el cual es capaz de soportar hasta 11 A. La resistencia de un cable AWG 20 se define por una distancia de 1 km según [31] como

$$R_{\mathrm{km}} = 33.2 \; \left[\frac{\Omega}{\mathrm{km}}\right].$$

Tabla 3.4: Máxima corriente y $\mathrm{P}_{R_{\mathrm{dif}}}$.

N_{ADMM}	Corriente máxima [A]	$\mathrm{P}_{R_{\mathrm{dif}}}\ \left[\frac{\mathrm{W}}{\Omega}\right]$
1	10.523	19.016
10	10.135	18.151
20	9.802	15.385
30	9.592	13.208
40	9.445	11.927
50	9.325	10.361

Tabla 3.5: THD de voltaje

	Lineal	**Saturado**
	quadprog	
	1.5999 %	1.5905 %
N_{ADMM}	**ADMM**	
1	1.5972 %	1.5912 %
10	1.5987 %	1.5931 %
20	1.5993 %	1.5934 %
30	1.5997 %	1.5928 %
40	1.5996 %	1.5923 %
50	1.5998 %	1.5918 %

Tabla 3.6: NRMSE: Diferencia entre estado estacionario y transiente en operación lineal y saturada.

	Lineal				**Saturado**			
	Transiente		**E. Estacionario**		**Transiente**		**E. Estacionario**	
N_{ADMM}	V_{c_d}	V_{c_q}	V_{c_d}	V_{c_q}	V_{c_d}	V_{c_q}	V_{c_d}	V_{c_q}
1	278.38 %	48.09 %	59.78 %	6.00 %	106.53 %	49.00 %	79.56 %	5.21 %
10	236.93 %	75.73 %	8.03 %	4.91 %	99.86 %	32.02 %	23.61 %	9.53 %
20	102.93 %	36.03 %	3.10 %	2.11 %	86.35 %	29.65 %	17.43 %	9.98 %
30	45.42 %	15.04 %	1.36 %	1.22 %	75.83 %	29.79 %	12.21 %	8.92 %
40	23.10 %	7.46 %	1.32 %	1.21 %	69.85 %	30.18 %	11.89 %	7.70 %
50	11.41 %	5.12 %	1.09 %	0.92 %	61.96 %	29.04 %	9.28 %	6.56 %

Ahora, si se toma como referencia un cableado por un total de 10 m de distancia

$$R = 33.2 \cdot 0.01 \ \ [\Omega]\,,$$

se tiene que la diferencia de potencia al utilizar ADMM en vez de quadprog sería

$$P_{dif_1} = \underbrace{19.016}_{\mathrm{P}_{R_{\mathrm{dif}_1}}} \cdot \underbrace{33.2 \cdot 0.01}_{R} = 6.3133 \ \ [\mathrm{W}]\,,$$

al tener sólo una iteración dentro del algoritmo ADMM, mientras que

$$P_{dif_{50}} = \underbrace{10.361}_{\mathrm{P}_{R_{\mathrm{dif}_{50}}}} \cdot \underbrace{33.2 \cdot 0.01}_{R} = 3.4399 \ \ [\mathrm{W}]$$

para cincuenta iteraciones en ADMM. Con ello se puede dilucidar que la diferencia de potencia es 1.8353 veces mayor para $N_{\mathrm{ADMM}} = 1$ en comparación con $N_{\mathrm{ADMM}} = 50$. Sin embargo, a medida que se tiene una mayor resistencia de línea o distancia de cable, mayor es el orden de magnitud de las pérdidas. De esta forma, se recomienda tener en cuenta las pérdidas adicionales para el dimensionamiento del cableado por una eventual sobrecorriente al utilizar ADMM.

3.3. Condiciones para implementación en hardware

La información proveída en este capítulo se resume en un sistema con las siguientes características para la implementación en hardware:

- A excepción del vector $\mathbf{c}$ del Algoritmo 1, todas las matrices y vectores de entrada son fijos.
- Se busca una implementación con un tiempo de ejecución menor al tiempo de muestreo (Δ) el cual es equivalente a 200 μs según se define en la Tabla 3.1.
- Se implementan seis casos con iteraciones fijas de ADMM según

$$N_{\text{ADMM}} \in \{1, 10, 20, 30, 40, 50\}.$$

- Se requiere de una etapa de procesamiento de mediciones previa a ADMM para definir las entradas al Algoritmo 1, el cual se basa en recalcular $\mathbf{c}$ para cada intervalo de control.
- El valor de ρ_0 en Algoritmo 1 es de 62.963413, el cual se obtiene mediante un análisis matemático al ejecutar la simulación del sistema de referencia utilizando el criterio mencionado en Selección de ρ (ver función *dhang_rho* en ADMM_QP.m).

4 Herramientas para desarrollo de algoritmo en FPGA

En este capítulo se explica el flujo de trabajo para la implementación en FPGA del algoritmo desarrollado donde cada sección corresponde a una de las herramientas utilizadas. Adicionalmente, existe una wiki[1] que contiene toda la información para replicar el proceso de diseño de hardware. Los códigos mencionados en la wiki se encuentran bajo acceso restringido a los investigadores del proyecto Fondecyt 1211676.

4.1. Vitis HLS

El software Vitis HLS es una herramienta de alto nivel que permite que funciones escritas en C, C++ y OpenCL puedan ser implementadas en un dispositivo de lógica programable [32]. La herramienta automatiza gran parte de las modificaciones de código requeridas para implementar y optimizar el código de alto nivel a lógica programable para lograr arquitecturas especializadas enfocadas en una solución con baja latencia o alto *throughput*. Además, a pesar de que existe la posibilidad de ingresar manualmente los pragmas, la herramienta es capaz de inferir algunos de ellos considerando tamaños de vectores, llamado a funciones, entre otros.

El flujo de trabajo en Vitis HLS consta de: [32]

- Compilar, simular y depurar el algoritmo descrito en C/C++.
- Ver los reportes para analizar y optimizar el diseño.
- Sintetizar el algoritmo en un diseño *Register-Transfer Level* (RTL) .
- Empaquetar la implementación en un objeto compilado de extensión .xo, o exportarlo a un RTL Intellectual Property (IP).

4.1.1. Optimizaciones de diseño a través de directivas o pragmas

Para definir optimizaciones de arquitectura, manejo de variables, interfaz de comunicación, entre otros, se pueden utilizar pragmas o directivas que indican requerimientos que la herramienta Vitis HLS interpreta como sugerencias. En caso de que Vitis HLS no logre cumplir estos requerimientos, la herramienta los omite y entrega un mensaje de advertencia informando la resolución del programa.

[1] `https://gitlab.com/jdjotad/ipd-500-juan-escarate/-/wikis/home`

Tabla 4.1: Ventajas y desventajas del uso de pragmas o directivas.

Formato	**Ventajas**	**Desventajas**
Archivos de directivas (Tcl Script)	Cada solución de Vitis HLS tiene directivas independientes. Este enfoque es ideal para exploraciones de diseño.	Si los archivos fuentes escritos en C son utilizados en una plataforma externa o archivados, el archivo *directives.tcl* debe ser incluido.
Código Fuente (Pragmas)	Las directivas de optimización se encuentran en el código fuente escrito en C. No se requiere de otros archivos para recrear los mismos resultados. Es un enfoque útil para directivas que son improbable que cambien.	Si la directiva de optimización está embebida en el código, es automáticamente aplicada a cada solución de Vitis HLS cuando se resintetiza.

Principalmente se pueden especificar optimizaciones en el diseño de RTL de dos formas: utilizando archivos de directivas, o incluyendo pragmas directamente en el código fuente [33] (ver Tabla 4.1).

En este trabajo se configura el proceso de compilación para poder paralelizar las operaciones matemáticas, ya que por defecto Vitis HLS mantiene el orden estricto de las operaciones cuando se sintetizan datos de tipo *float* y *double*. Esta restricción puede ser anulada utilizando el flag *unsafe_math_optimizations* en las configuraciones de compilación [34], la cual se realiza con el siguiente comando:

```
config_compile -unsafe_math_optimizations=true
```

De esta forma, se ignora el signo del valor cero en punto flotante y permite operaciones asociativas de manera que el compilador pueda realizar optimizaciones agresivas en operaciones de punto flotante. El uso de esta opción puede cambiar el resultado de algún cálculo y ocasionar que el resultado no concuerde con la simulación en C/RTL por lo que hay que asegurarse que la aplicación sea tolerante a diferencias marginales [35].

4.1.2. Flujo de trabajo

La creación del proyecto, inclusión de códigos fuente, configuraciones de compilación y un enlace a los códigos utilizados en esta tesis se encuentran en la sección Vitis HLS[2] de la wiki del repositorio.

Luego de obtener un bloque IP de la herramienta Vitis HLS, se utiliza este resultado para implementar un diseño de bloques en Vivado, el cual entregará directamente un archivo para programar los bloques lógicos de la FPGA.

[2] https://gitlab.com/jdjotad/ipd-500-juan-escarate/-/wikis/VitisHLS

4.2. Vivado

Vivado es un software para síntesis y análisis de *Hardware Description Language*. En este contexto, la herramienta ofrece múltiples formas de desempeñar las funciones de diseño, implementación y verificación de los dispositivos de AMD Xilinx. Se puede utilizar el flujo de diseño FPGA *Register Transfer Level* (RTL)-*to*-bitstream. También, existe otra forma de trabajo que consiste en flujo de integración a nivel de sistema, el cual se enfoca en un diseño centrado en el uso de IPs y basado en C [36]. En este trabajo se utiliza el enfoque centrado en el uso de IPs mediante una descripción por diagrama de bloques.

4.2.1. Flujo de trabajo

Para una descripción de los pasos para replicar el diagrama de bloques en Vivado utilizado en esta tesis referirse al repositorio al índice Vivado[3] en la wiki del repositorio.

Una vez se realice la configuración de cada uno de los bloques del diseño se obtiene un diagrama de bloques como se muestra en la Figura 4.1. Luego se selecciona la opción *Generate Bitstream* para obtener una archivo bitstream con el que se procederá a programar la FPGA mediante el uso de PYNQ.

4.3. PYNQ

PYNQ es un proyecto de código abierto desarrollado por AMD el cual provee un entorno de trabajo basado en Jupyter Notebook con APIs de Python para utilizar plataformas de cómputo adaptativas AMD Xilinx [37].

La lógica programable resultante de un proyecto en Vivado se presenta como una librería de hardware llamada *overlay*. Estos *overlays* pueden ser accedidos a través de una API de Python. Además, con PYNQ se puede programar el procesador embebido. El entorno de trabajo incorpora una infraestructura de Jupyter Notebook que corre un kernel de Python Interactivo (IPython).

4.3.1. Flujo de trabajo

Los requerimientos para lograr implementar esta última etapa se presentan en el índice PYNQ[4] de la wiki del repositorio, donde se incluye tanto un código de ejemplo como un enlace hacia los códigos utilizados para obtener los resultados de esta tesis.

Al tener una espacio de trabajo como Jupyter Notebook, se realizan múltiples tareas como programación de bloques lógicos en FPGA, manejo de datos entre CPU y FPGA, verificación de funcionamiento, y medición de tiempo de ejecución.

[3] `https://gitlab.com/jdjotad/ipd-500-juan-escarate/-/wikis/Vivado`
[4] `https://gitlab.com/jdjotad/ipd-500-juan-escarate/-/wikis/PYNQ`

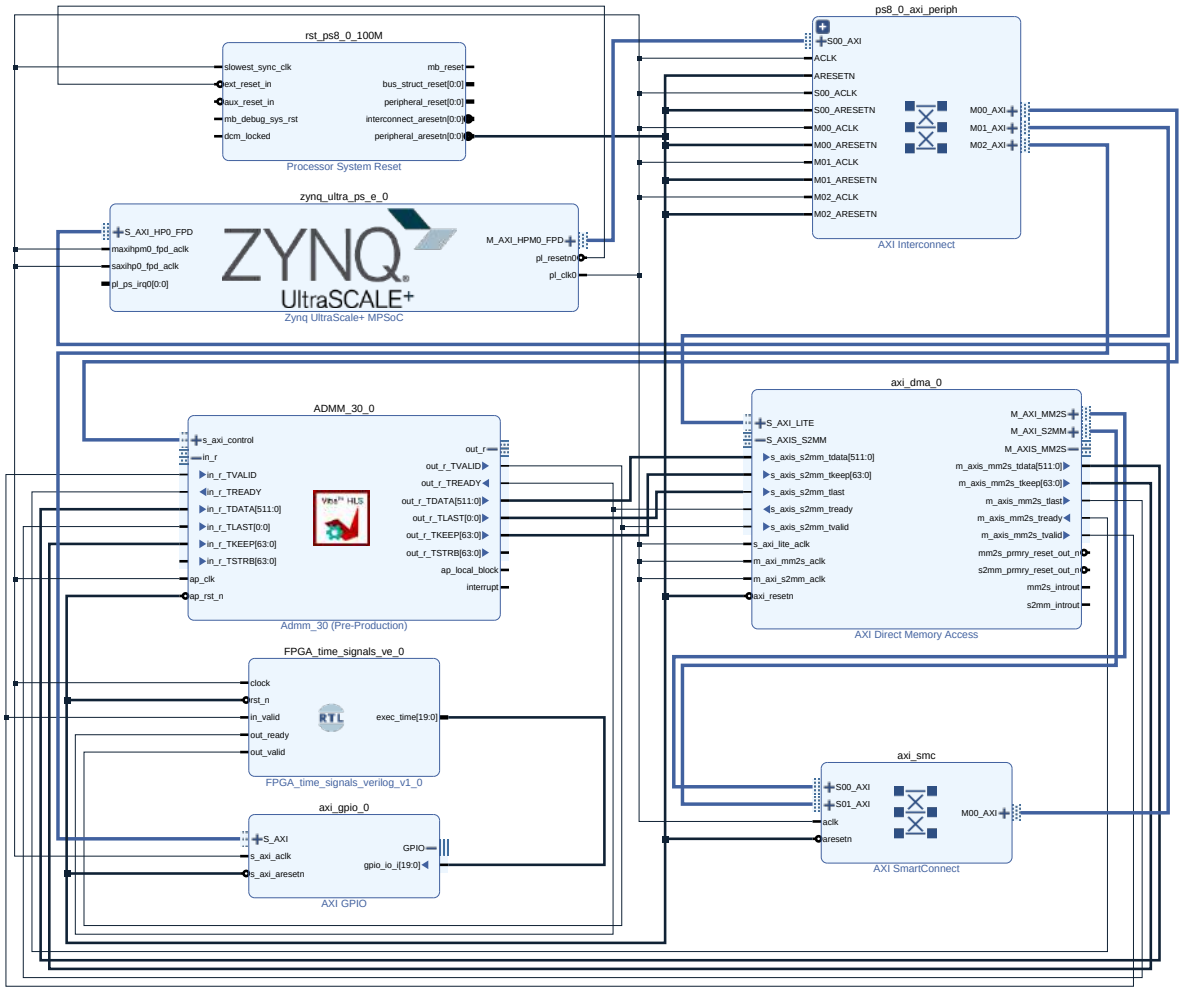


Figura 4.1: Diagrama de bloque final en Vivado.

5 | Implementación en CPU y FPGA

Para efectos de esta sección, una implementación en software es equivalente a una implementación en CPU compilando un código descrito en C++ . Por otra parte, una implementación en hardware es equivalente al resultado del flujo de diseño presentado en el Capítulo 4, para implementar en FPGA.

La plataforma de desarrollo que se utiliza en este trabajo es la ZCU104. Esta tarjeta se encuentra en la categoría de MPSoC, la cual contiene una CPU y FPGA embebidas, además de una variedad de periféricos. Por consiguiente, la ejecución en CPU y FPGA se realizan en la misma plataforma. En particular, en este trabajo se utiliza el compilador g++ y la CPU embebida en la tarjeta ZCU104 para la implementación en software, mientras que la implementación en hardware se realiza en la FPGA de la misma.

Para cada una de las dos implementaciones realizadas en esta tesis se muestran los pragmas utilizados, el uso de recursos de FPGA, y los tiempos de ejecución en FPGA y CPU.

Los códigos y la metodología para replicar los resultados se encuentran en la carpeta ZCU104[1] del repositorio. Los códigos fuente son parte de un proyecto de investigación que se encuentra en curso al momento de terminar este trabajo de tesis, por lo que su acceso se encuentra restringido a los investigadores asociados al proyecto Fondecyt 1211676. Se espera que en el futuro estos códigos y las instrucciones para su uso queden disponibles públicamente.

5.1. Metodología para implementación en CPU y FPGA

La implementación comienza con una etapa de simulación en un computador de escritorio (PC) utilizando MATLAB/Simulink, donde se almacena un archivo con las señales de interés, el cual se debe guardar en un directorio dentro de la tarjeta ZCU104. Luego, se lleva a cabo el proceso de diseño presentado en el Capítulo 4, obteniendo dos archivos que se deben almacenar en la tarjeta ZCU104: un archivo *bitstream* (.bit) y un archivo *handoff* (.hwh).

Para la implementación en software se utilizan los mismos códigos descritos en C++ para la etapa de diseño de Vitis HLS presentado en el Capítulo 4 que se pueden encontrar en la carpeta *cpp_files* del repositorio. Mientras que la implementación en hardware utiliza los archivos bitstream y handoff que se encuentran en las carpetas *bitfiles_ ADMM* y *bitfiles_ CCS-MPC*.

La metodología para la implementación en software dentro de la ZCU104 consta de los siguientes pasos:

[1] `https://gitlab.com/jdjotad/ipd-500_juan_escarate/-/tree/main/ZCU104`

1. Cargar los valores de simulación y códigos fuente.
2. Compilar con g++.
3. Correr los ejecutables generados con el compilador y almacenar tanto la actuación resultante del algoritmo, como el tiempo de ejecución.
4. Verificar funcionalidad comparando los resultados de la implementación en CPU con los valores simulados.
5. Si el error entre la simulación y la implementación en software es cercana a 0 %, se almacenan y reportan los tiempos de ejecución.

Para la implementación en hardware dentro de la tarjeta ZCU104 se realizan los siguientes pasos:

1. Cargar los valores de simulación y archivos bitstream y handoff.
2. Programar la FPGA mediante el uso del paquete PYNQ utilizando los archivos bitstream y handoff.
3. Transmitir las entradas al algoritmo desde la CPU hacia la FPGA a través del bloque DMA[2], el cual se llama "*AXI Direct Memory Access*" en el diagrama de bloques presentado en la Figura 4.1.
4. Esperar el cómputo en hardware, para luego recibir los resultados desde la FPGA hacia la CPU a través del bloque DMA.
5. Verificar funcionalidad comparando los resultados de la implementación en FPGA con los valores simulados.
6. Si el error entre la simulación y la implementación en hardware es cercana a 0 %, se almacenan y reportan los tiempos de ejecución.

La metodología previamente mencionada se puede visualizar de manera gráfica en la Figura 5.1, donde los nombres de los archivos son representativos.

Una vez explicada la metodología para tanto la implementación en software como en hardware, se muestra el código fuente en C++ que describe el *solver* de optimización ADMM con iteraciones fijas.

5.2. Descripción de ADMM en C++

El solver de optimización ADMM se describe en C++ (ver Código 5.1) basándose en el Algoritmo 1. Dentro del código se incluye un requerimiento a través del uso de pragmas, donde se sugiere a la herramienta Vitis HLS la explotación de recursos para crear pipeline cuando sea posible y con el menor intervalo de iniciación posible. Cabe destacar que cada una de las funciones *x_ min*, *z_ min* y *u_ update* detallan las operaciones matemáticas (2.14), (2.16), y (2.17) respectivamente, y se pueden encontrar dentro del archivo *utils.hpp*. Además, la implementación en software utiliza el mismo código sin el uso de pragmas.

[2] *Direct memory access* (DMA) es una característica de sistemas de cómputo que permite que ciertos subsistemas de hardware accedan al sistema de memoria principal de manera independiente de la CPU.

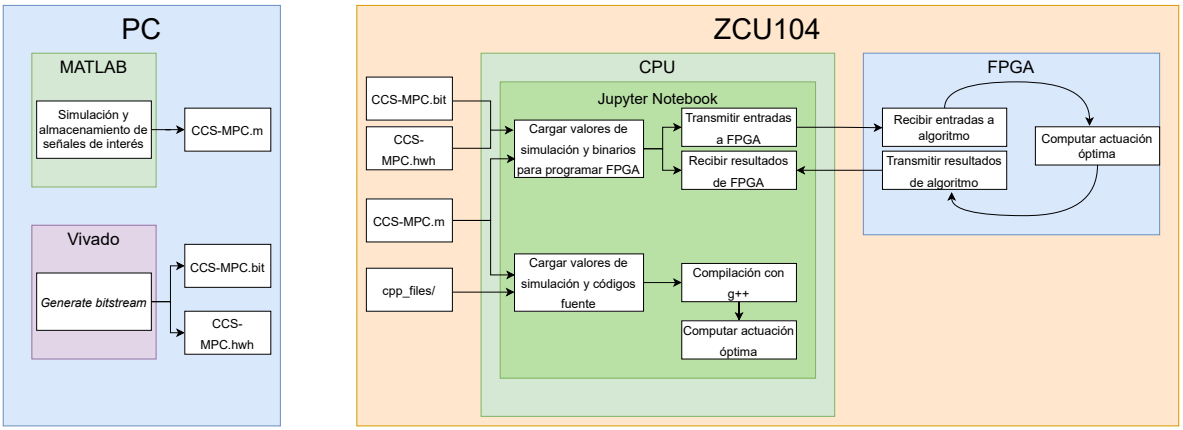


Figura 5.1: Representación gráfica de metodología para implementación en CPU y FPGA.

Las entradas al Código 5.1 se relacionan con el Algoritmo 1 de la siguiente forma:

- $\mathbf{Q}$: No se requiere dado que sólo influye en el cálculo de $\mathbf{R}_{\text{inv}}$.
- $\mathbf{q}$: Sólo se utiliza el vector negado de q, por lo que no se considera esta entrada.
- $\mathbf{A}$: Se utiliza como A.
- $\mathbf{c}$: Se utiliza como c.
- N_{ADMM}: Se renombra por el parámetro IT en el *template* de la función.
- $\mathbf{x_0}$: Se renombra como x.
- $\mathbf{z_0}$: Se renombra como z.
- $\mathbf{u_0}$: Se renombra como u.
- ρ_0: Se renombra como ρ.

Además de las entradas del Algoritmo 1, se incluyen los siguientes valores:

- q_neg: Equivale a $-\mathbf{q}$.
- R_inv: Equivale a $\mathbf{R}_{\text{inv}}$. Debido a que $\mathbf{Q}$, ρ y $\mathbf{A}$ son fijos, se precalcula $\mathbf{R}_{\text{inv}}$ (ver Algoritmo 1).
- rho_neg_A_T: Equivale a $-\rho\mathbf{A}^{\text{T}}$
- A_neg: Equivale a $-\mathbf{A}$
- c_neg: Equivale a $-\mathbf{c}$

Es importante destacar que en el Código 5.1, las únicas entradas que varían en cada paso de la simulación son c y c_neg. Por lo tanto, todas las otras entradas son almacenadas en memoria.

Código 5.1: Descripción de ADMM en C++ con pragma de pipeline para Vitis HLS.

```
template<int IT, int N, int M, int P, typename T>
void ADMM(T (&q_neg)[N][P], T (&rho_neg_A_T)[N][M], T (&A)[M][N],
          T (&A_neg)[M][N], T (&c)[M][P],           T (&c_neg)[M][P],
          T (&R_inv)[N][N], T (&x)[N][P],           T (&z)[M][P],
          T (&u)[M][P]){
    // Optimization iterations
    for(int i = 0; i < IT; i++){
#pragma HLS PIPELINE
        x_min(q_neg, rho_neg_A_T, c_neg, R_inv, x, z, u);
        z_min(A_neg, c, x, z, u);
        u_update(A, c_neg, x, z, u);
    }
}
```

5.3. Implementación de ADMM en CPU y FPGA

Esta implementación sólo lleva a cabo el cómputo del problema de optimización en software o hardware, lo que significa que se diseña una implementación de ADMM. La descripción del *solver* ADMM en C++ se muestra en el Código 5.1.

Dado que internamente el controlador CCS-MPC propuesto consta de dos etapas (ver Figura 5.2), existe una etapa de procesamiento obligatoria, que para efectos de esta implementación, se desarrolla en la CPU embebida en la tarjeta de desarrollo, ya que los valores se obtienen del archivo proveniente de la simulación en MATLAB/Simulink. Es por esto que la implementación en FPGA requiere de comunicación con la CPU para implementar el controlador. A causa de la necesidad de transmisión de datos entre CPU y FPGA para utilizar este esquema de control, se presentarán los resultados de tiempo de ejecución en hardware de la siguiente forma:

- FPGA + Comunicación: Este tiempo incluye los siguientes tres tiempos de ejecución:
  1. Transmisión de datos de entrada al algoritmo ADMM desde CPU a FPGA.
  2. Ejecución de algoritmo ADMM en FPGA.
  3. Transmisión de datos de salida del algoritmo ADMM desde FPGA a CPU.
- FPGA: Tiempo que demora en ejecutarse el algoritmo ADMM en FPGA.

Previo al almacenamiento de los tiempos de ejecución, existe una etapa de verificación de resultados tanto para CPU como FPGA. Una vez que se obtiene una raíz del error cuadrático medio normalizada (NRMSE) cercano al 0 % para cada estado, se crea una tabla con los tiempos de ejecución. Los tiempos de ejecución son calculados de la siguiente forma:

- Software/CPU
  - Uso del *header chrono*: Se almacena el tiempo de ejecución de la siguiente forma:

    ```
    #include <chrono>
    ```

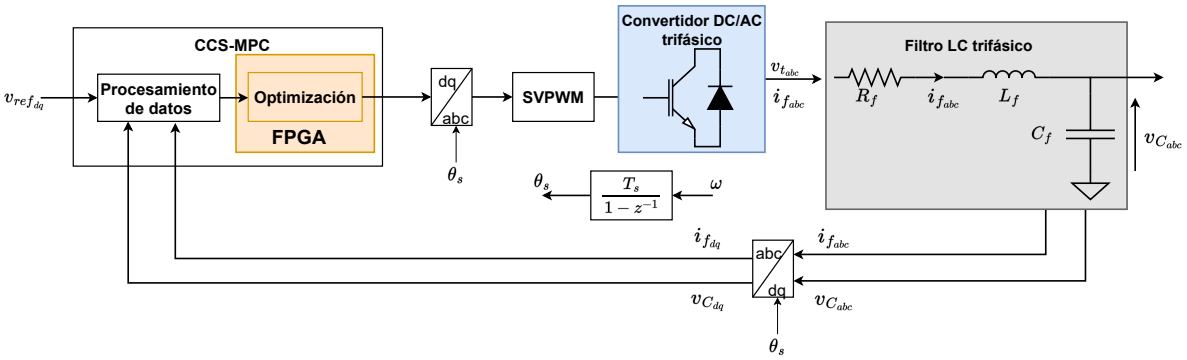


Figura 5.2: Representación gráfica de implementación de ADMM en FPGA.

```
#include <stdlib.h>

int main(){
    auto start = std::chrono::high_resolution_clock::now();
    test_function();
    auto end   = std::chrono::high_resolution_clock::now();

    auto time = std::chrono::duration_cast<std::chrono::nanoseconds>(
    end - start).count();
}
```

De esta forma la variable *time* contiene un valor entero de la cantidad de nanosegundos que demora la función *test_function*.

- Hardware/FPGA
  - Uso de *timeit* de python para tiempo "FPGA + Comunicación": El uso de esta función se emplea de la siguiente forma

    ```
    import timeit

    time = timeit.timeit(lambda: test_function(), number=1)
    ```

    Con esto se tiene la variable *time* representa el tiempo en segundos que demora en correr la función *test_function*, que en nuestro caso se encarga computar en hardware, además de enviar y recibir datos entre CPU y FPGA.

  - Lectura de señal *exec_time* del módulo *FPGA_time_signals* para tiempo "FPGA": Este módulo se describe utilizando Verilog y se implementa en hardware dentro de la arquitectura diseñada a través del diagrama de bloques en Vivado. La salida de este módulo indica el tiempo que demora la FPGA en computar la salida de la implementación de ADMM. En este contexto, la señal *exec_time* contiene la cantidad de ciclos de reloj, el cual está configurado en 10 ns.

La Tabla 5.1 presenta los tiempos de ejecución en µs para CPU y FPGA. La transmisión desde CPU a FPGA es de 74 datos tipo flotante y desde FPGA a CPU de 16 datos de tipo flotante. Además, la implementación se basa en una simulación de 0.5 s con un período de muestreo de 200µs, lo que equivale a un total de 2500 muestras. En CPU, se tiene únicamente un

Tabla 5.1: Tiempo de ejecución en µs para implementación de ADMM en CPU y FPGA.

	CPU			**FPGA + Comunicación**			**FPGA**
IT	**Promedio**	**Máximo**	**Mínimo**	**Promedio**	**Máximo**	**Mínimo**	
1	176	295	175	359	587	353	5
10	1747	2654	1741	361	1294	355	45
20	3494	4733	3481	363	913	357	88
30	5247	6665	5234	366	946	359	132
40	6984	8301	6967	390	1849	382	175
50	8731	10056	8715	436	701	426	219

caso que logra una implementación viable dentro del requerimiento de tiempo de la aplicación de 200 µs. Además, todas las implementaciones en software con más de una iteración superan 1 ms, lo que hace inviable una implementación utilizando la CPU embebida de la tarjeta ZCU104. Por otra parte, el cómputo que se lleva a cabo en la FPGA sólo supera los 200 µs para 50 iteraciones. Sin embargo, dado que el controlador requiere de una etapa de procesamiento de datos, además del algoritmo ADMM, existe un overhead asociado a la comunicación entre FPGA y CPU. Dado que la tarjeta de desarrollo ZCU104 utiliza un sistema operativo Linux junto al uso de PYNQ, el tiempo de comunicación varía según el *scheduler*, lo que no es controlable a nivel usuario.

Gráficamente, se puede visualizar en la Figura 5.3 que el tiempo de ejecución del algoritmo ADMM en FPGA presenta un comportamiento lineal con respecto a la cantidad de iteraciones. Adicionalmente, el tiempo promedio de comunicación varía entre 200 µs y 350 µs. Es por esto que, a pesar de obtener resultados favorables para la implementación de ADMM en FPGA, se debe incluir la etapa de procesamiento en la FPGA, apuntando a eliminar la transmisión entre CPU y FPGA como parte del control.

El uso de recursos que arroja tanto la herramienta de diseño Vitis HLS y Vivado se muestran en la Tabla 5.4. Para una iteración, se tiene que el uso de recursos es mucho mayor que los otros casos, incluso, el uso de DSP que arroja Vitis HLS supera el disponible. Esto se debe principalmente a que el pragma de pipeline se encuentra en un proceso no iterativo. Sin embargo, la herramienta Vivado es capaz de disminuir el uso de DSPs a 1688 de manera autónoma, logrando un diseño funcional. Los casos con una cantidad de iteraciones por sobre uno presentan un uso de recursos similar, ya que el llamado de las funciones en el proceso iterativo se realiza por defecto con el pragma *inline off*, ocasionando que las funciones se implementen como módulos que se llaman de manera iterativa. Además, existen diferencias entre Vitis HLS y Vivado en el uso de LUT, FF y BRAM, donde ocurren principalmente dos fenómenos: el uso de BRAM que arroja Vitis HLS es parcialmente reemplazado por LUTRAM en la implementación final en Vivado, y la cantidad de LUT y FF es mayor en la implementación final de Vivado que en las estimaciones de Vitis HLS.

Dado que la comunicación entre CPU y FPGA agrega un *overhead* que es del orden de magnitud del requerimiento de tiempo, se crea una implementación que evita el uso de CPU dentro del controlador. De esta forma, se simula el sensado de señales en la CPU, donde esta lee un archivo con los datos almacenados de MATLAB y los envía a la FPGA para luego medir la salida del sistema. A pesar de que existe una comunicación involucrada, el tiempo que demora

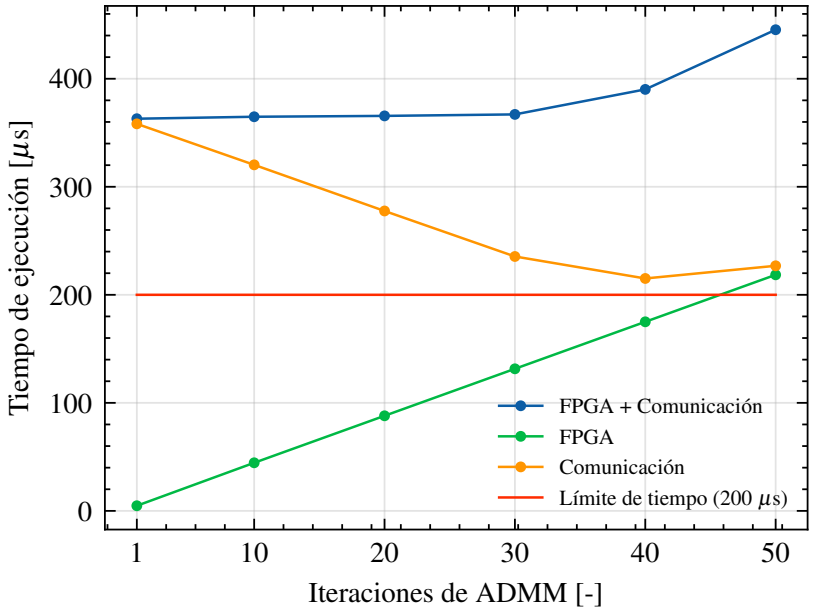


Figura 5.3: Gráfico de tiempo de ejecución (promedio) vs iteraciones de ADMM en FPGA.

Tabla 5.2: Uso de recursos para implementación de ADMM en FPGA.

IT		LUT	LUTRAM	FF	BRAM	DSP	BUFG
1	Vitis HLS	176 157	-	170 427	2056	2 084	-
	Vivado	129 929	2 226	132 006	176	1 688	13
10	Vitis HLS	70 698	-	61 761	264	320	-
	Vivado	82 160	2 226	70 698	88	320	8
20	Vitis HLS	70 698	-	61 762	264	320	-
	Vivado	82 170	2 226	80 559	88	320	9
30	Vitis HLS	70 698	-	61 762	264	320	-
	Vivado	82 155	2 226	80 540	88	320	9
40	Vitis HLS	70 700	-	61 763	264	320	-
	Vivado	82 220	2 226	80 573	88	320	8
50	Vitis HLS	70 700	-	61 763	264	320	-
	Vivado	82 196	2 226	80 555	88	320	9
Disponible		230 400	101 760	460 800	312	1 728	544

la transferencia de datos simula un sensado, por lo que no aporta al tiempo de ejecución del controlador.

5.4. Implementación de lazo de control CCS-MPC en FPGA

La representación gráfica presentada en Figura 5.4 muestra la forma en que se implementa el lazo completo de CCS-MPC en FPGA. De esta forma, se pasa primero por una etapa de procesamiento de la información de sensado y el valor de referencia para el voltaje de salida. Luego, el vector de salida de este bloque junto con la lectura de matrices y vectores

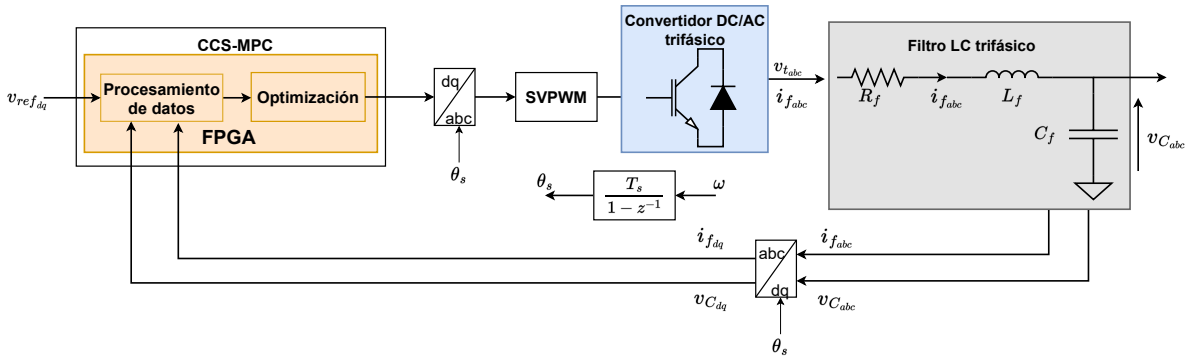


Figura 5.4: Representación gráfica de implementación de lazo de control CCS-MPC en FPGA.

Tabla 5.3: Tiempo de ejecución en µs para implementación de CCS-MPC en CPU y FPGA.

	CPU			FPGA
IT	**Promedio**	**Máximo**	**Mínimo**	
1	182	344	181	6
10	1754	3114	1747	46
20	3499	4854	3487	106

precalculados en FPGA, se utilizan como entradas para el algoritmo de ADMM.

A pesar de que el código de ADMM es exactamente el mismo, la localización del pragma para indicar pipeline se cambia de posición. Esta decisión se lleva a cabo debido a que al agregar la etapa de procesamiento en la descripción en C++ , Vitis HLS arroja un error de *timing* y al probar la implementación obtenida de Vivado, el resultado que arroja el solver es indefinido (Nan), por lo que la implementación no es capaz de describir el comportamiento deseado en el período de reloj definido. Es por esto que, se prueban distintas variaciones del uso de pragmas para disminuir el tiempo de ejecución, logrando resolver el problema de timing con la utilización del pragma pipeline fuera del proceso iterativo (ver Código 5.2). Sin embargo, con la ubicación de este pragma, la arquitectura resultante no presenta un uso de recursos constante, por lo que la herramienta de Vivado sólo permite implementar hasta veinte iteraciones. Al procesar la salida de Vitis HLS con treinta iteraciones de ADMM en Vivado, este no finaliza el proceso de síntesis de manera satisfactoria al trabajar con la versión 2021.2 en Linux, y 2022.1 en Windows arrojando errores asociados al uso de RAM para Linux y de síntesis para Windows.

Para más información del código empleado revisar la función *preprocessing_ c* del archivo *utils.hpp* y su llamado previo a ADMM (ver línea 39 de *ADMM_ wrapper.cpp*). En particular, la salida de *preprocessing_ c*, que es el vector **c**, se utiliza para definir las entradas c y c_neg del optimizador ADMM. Todos los códigos fuente en C++ se encuentran dentro de la carpeta *cpp_ files/CCS-MPC*.

Los tiempos de ejecución de la implementación del lazo CCS-MPC en FPGA se muestran en la Tabla 5.3, donde al igual que la implementación anterior se presenta un total de 2500 muestras bajo las mismas condiciones de simulación. Se tiene un aumento en el tiempo de ejecución de 1 µs, 1 µs y 18 µs para las implementaciones de ADMM con 1, 10, y 20 iteraciones respectivamente. Además, tanto los tiempos de CPU como en FPGA son similares a la implementación anterior,

Código 5.2: Descripción de ADMM en C++ con pragma de pipeline fuera de proceso iterativo para Vitis HLS.

```
template<int IT, int N, int M, int P, typename T>
void ADMM(T (&q_neg)[N][P], T (&rho_neg_A_T)[N][M], T (&A)[M][N],
          T (&A_neg)[M][N], T (&c)[M][P],           T (&c_neg)[M][P],
          T (&R_inv)[N][N], T (&x)[N][P],           T (&z)[M][P],
          T (&u)[M][P]){
#pragma HLS PIPELINE
    // Optimization iterations
    for(int i = 0; i < IT; i++){
        x_min(q_neg, rho_neg_A_T, c_neg, R_inv, x, z, u);
        z_min(A_neg, c, x, z, u);
        u_update(A, c_neg, x, z, u);
    }
}
```

Tabla 5.4: Uso de recursos para implementación de CCS-MPC en FPGA.

IT		LUT	LUTRAM	FF	BRAM	DSP	BUFG
1	Vitis HLS	78 722	-	63 643	264	320	-
	Vivado	83 842	2 071	82 479	92	320	8
10	Vitis HLS	88 584	-	75 060	318	395	-
	Vivado	92 031	2 121	94 294	119	395	9
20	Vitis HLS	114 482	-	110 675	302	551	-
	Vivado	130 705	2 211	126 945	111	551	10
Disponible		230 400	101 760	460 800	312	1 728	544

por lo que se verifica que el costo computacional se debe principalmente al cómputo de la solución al problema de optimización. Las tres implementaciones en FPGA respetan las restricciones de tiempo real impuestas por la aplicación, demostrando evidencia del potencial del uso de FPGAs para la implementación de esquemas de control avanzados.

El uso de recursos (ver Tabla 5.4) aumenta según la cantidad de iteraciones, diferencia que se debe a la ubicación del *pragma* de *pipeline*, y ocasiona que este diseño no sea escalable por sobre las veinte iteraciones de ADMM. Para realizar un estudio hasta cincuenta iteraciones, se requiere de una etapa de exploración de pragmas más prolongada.

6 | Conclusiones y trabajo futuro

En este trabajo se implementa en software y hardware el algoritmo ADMM como *solver* de optimización para un esquema de control CCS-MPC en una aplicación de electrónica de potencia. Inicialmente, una simulación en MATLAB/Simulink permite medir el desempeño del controlador al reemplazar un método de referencia (*quadprog*) por la implementación de ADMM planteada en este trabajo. Luego de simular y definir los casos de estudio, se utiliza un código en C++ para definir el comportamiento del *solver* tanto para CPU como FPGA. Mientras en CPU se requiere el uso de un compilador como g++, el flujo de diseño para FPGA requiere del uso de las herramientas Vitis HLS y Vivado de AMD Xilinx. Adicionalmente, las pruebas para programar la FPGA y corroborar el funcionamiento se realizan mediante el uso de PYNQ, un proyecto de AMD Xilinx.

En este capítulo se presentan los principales resultados y se plantean ideas para trabajo futuro.

6.1. Conclusiones

Luego de una introducción a elementos de electrónica de potencia y control, asociados a la aplicación de referencia, se realiza un análisis de desempeño de los casos a implementar en FPGA. Dado que el objetivo principal es regular la magnitud y frecuencia del voltaje de salida, es relevante el valor de THD para cada caso de ADMM, el cual indica la distorsión armónica de la señal resultante en estado estacionario. Debido a que el THD es menor a 1.6 % tanto al utilizar el solver de referencia *quadprog* como los seis casos de prueba al utilizar el solver ADMM, se procede a analizar el fenómeno de sobrecorriente que se presenta en estado transiente al utilizar ADMM. Con ello, se tiene una referencia visual del comportamiento dinámico de la respuesta del sistema con ADMM, además de la diferencia de potencia al utilizar ADMM en vez de quadprog, la cual varía entre 19.016 $\left[\frac{\mathrm{W}}{\Omega}\right]$ y 10.361 $\left[\frac{\mathrm{W}}{\Omega}\right]$, para una y cincuenta iteraciones de ADMM respectivamente. Con esta información se procede a la etapa de diseño e implementación en hardware para medir el tiempo de ejecución en FPGA.

Con los pasos que se muestran en el flujo de diseño presentado en el Capítulo 4, se implementan los dos enfoques reportados en este documento. Uno de ellos consta de la implementación del *solver* ADMM, mientras que el otro considera la etapa de procesamiento de datos, de manera que describa el controlador en su totalidad. El tiempo de ejecución se presenta tanto en CPU como en FPGA, utilizando la plataforma de desarrollo ZCU104, la cual contiene CPU *multicore*, FPGA y múltiples periféricos.

El primer enfoque consta de una implementación del *solver* de optimización ADMM en

FPGA. La implementación en software no presenta resultados favorables para los requisitos temporales de esta aplicación. Sin embargo, la implementación en FPGA logra un tiempo de ejecución conforme a los requerimientos de tiempo real de la aplicación (200 µs) hasta cuarenta iteraciones de ADMM, sin considerar la transmisión de datos. A pesar de ello, este enfoque requiere de la transmisión de datos como parte del proceso de control, lo cual le agrega un *overhead* de al menos 200 µs para cada uno de los casos de estudio. De esta forma se concluye que, a pesar de obtener una implementación en hardware que es capaz de computar la solución óptima del problema de optimización planteado para la aplicación en cuestión, incluir una etapa de transmisión CPU-FPGA como parte del controlador ocasiona que el esquema de control no cumpla con los requisitos temporales impuestos por el período de muestreo.

El segundo enfoque consta de una implementación de la etapa de control, que incluye tanto el procesamiento de las mediciones junto con la referencia, como la resolución del problema de optimización utilizando el *solver* ADMM. Nuevamente, la implementación en software es de utilizada para reportar la aceleración al implementar la solución en FPGA, dado que los tiempos de ejecución superan los 200 µs. Por otra parte, la implementación del controlador en hardware no requiere de transmisión CPU-FPGA, la transferencia de datos se utiliza únicamente para simular el sensado de señales, de manera que no afecta el tiempo de ejecución del controlador. Con este enfoque se logra implementar en hardware los casos para $N_{\text{ADMM}} \in \{1, 10, 20\}$, logrando computar una solución al problema de optimización dentro de los 200 µs.

En conclusión, con este trabajo se logra validar el uso de herramientas de alto nivel para el diseño e implementación del *solver* de optimización ADMM en FPGAs, describiendo en hardware la etapa de control en una aplicación de electrónica de potencia. Además de presentar un esquema de control capaz de actuar respetando límites de operación, se obtiene una solución determinista que cumple con los requisitos temporales de la aplicación, demostrando la capacidad de una implementación en hardware a través del uso de FPGAs para la descripción de esquemas de control predictivos con cálculos *online*.

6.2. Trabajo futuro

En este trabajo se reconocen dos enfoques principales para trabajo futuro:

- Exploración avanzada de pragmas y uso de datos en punto fijo: En este trabajo se realizó una etapa de exploración de pragmas que no se encuentra reportada debido a que no era el enfoque principal de investigación. Sin embargo, teniendo una implementación funcional, se recomienda explorar distintas variaciones de implementación al utilizar pragmas y evaluar el uso de un tipo de datos descrito en punto fijo.
- Diseño de álgebra matemática tipo *sparse*: Debido a la existencia de matrices tipo *sparse* dentro del sistema, la creación de funciones que logren implementar álgebra explotando esta característica, lograría disminuir el tiempo de ejecución.

Bibliografía

[1] X. Fang, S. Misra, G. Xue, and D. Yang, "Smart grid—the new and improved power grid: A survey," *IEEE communications surveys & tutorials*, vol. 14, no. 4, pp. 944–980, 2011.

[2] Y. P. Adam Hirsch and J. Guerrero, "Microgrids: A review of technologies, key drivers, and outstanding issues," *Renewable and Sustainable Energy Reviews*, vol. 90, pp. 402–411, 2018.

[3] E. Hossain, E. Kabalcı, R. Bayindir, and R. Perez, "A comprehensive study on microgrid technology," *International Journal of Renewable Energy Research*, vol. 4, pp. 1094–1104, 01 2014.

[4] J. R. Lopez, P. P. Cruz, and A. M. Gutierrez, "Vsi lc filter optimized by a genetic algorithm from connected to island microgrid operation," *Energy Systems*, pp. 1–19, 2021.

[5] G. A. Papafotiou, G. D. Demetriades, and V. G. Agelidis, "Technology readiness assessment of model predictive control in medium-and high-voltage power electronics," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 9, pp. 5807–5815, 2016.

[6] S. Vazquez, J. Rodriguez, M. Rivera, L. G. Franquelo, and M. Norambuena, "Model predictive control for power converters and drives: Advances and trends," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 2, pp. 935–947, 2017.

[7] C. Bordons, F. Garcia-Torres, and M. A. Ridao, *Model predictive control of microgrids*. Springer, 2020, vol. 358.

[8] P. Harisyam, V. Prasanth, V. Natarajan, and K. Basu, "Continuous control set model predictive control of buck converter," in *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, 2020, pp. 1297–1302.

[9] Z. Zhang, O. Babayomi, T. Dragicevic, R. Heydari, C. Garcia, J. Rodriguez, and R. Kennel, "Advances and opportunities in the model predictive control of microgrids: Part i–primary layer," *International Journal of Electrical Power Energy Systems*, vol. 134, p. 107411, 2022.

[10] M. Preindl and S. Bolognani, "Comparison of direct and pwm model predictive control for power electronic and drive systems," in *IEEE Applied Power Electronics Conference and Exposition*, 2013, pp. 2526–2533.

[11] R. Pérez-Ibacache, A. L. Cedeño, C. A. Silva, G. Carvajal, J. C. Agüero, and A. Yazdani, "Decentralized model-based predictive control for der units integration in ac microgrids

subject to operational and safety constraints," *IEEE Transactions on Power Delivery*, vol. 36, no. 4, pp. 2479–2489, 2021.

[12] Reinier Lopez, "Control Predictivo Basado en Modelo para un inversor con un filtro LC a la salida en Recursos Energéticos Distribuidos (DERs)," Master's thesis, Universidad Técnica Federico Santa María, Chile, 2021.

[13] K. Ling, B. Wu, and J. Maciejowski, "Embedded model predictive control (mpc) using a fpga," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 15 250–15 255, 2008, 17th IFAC World Congress.

[14] S. Lucia, D. Navarro, Lucía, P. Zometa, and R. Findeisen, "Optimized fpga implementation of model predictive control for embedded systems using high-level synthesis tool," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 1, pp. 137–145, 2018.

[15] I. McInerney, G. A. Constantinides, and E. C. Kerrigan, "A survey of the implementation of linear model predictive control on fpgas," *IFAC-PapersOnLine*, vol. 51, no. 20, pp. 381–387, 2018.

[16] N. Komodakis and J.-C. Pesquet, "Playing with duality: An overview of recent primal?dual approaches for solving large-scale optimization problems," *IEEE Signal Processing Magazine*, vol. 32, no. 6, pp. 31–54, 2015.

[17] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari, "Embedded online optimization for model predictive control at megahertz rates," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3238–3251, 2014.

[18] T. V. Dang, K. Ling, and J. Maciejowski, "Embedded admm-based qp solver for mpc with polytopic constraints," in *2015 European Control Conference (ECC)*, 2015, pp. 3446–3451.

[19] G. Banjac, B. Stellato, N. Moehle, P. Goulart, A. Bemporad, and S. Boyd, "Embedded code generation using the osqp solver," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, 2017, pp. 1906–1911.

[20] P. Zhang, J. Zambreno, and P. H. Jones, "An embedded scalable linear model predictive hardware-based controller using admm," in *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2017, pp. 176–183.

[21] M. Schubiger, G. Banjac, and J. Lygeros, "Gpu acceleration of admm for large-scale quadratic programming," *Journal of Parallel and Distributed Computing*, vol. 144, p. 55–67, Oct 2020.

[22] N. Mohan, T. M. Undeland, and W. P. Robbins, *Power electronics: converters, applications, and design.* John wiley & sons, 2003.

[23] K. Bimal, *Modern power electronics and AC drives.* Prentice-Hall, 2001.

[24] K. V. Kumar, P. A. Michael, J. P. John, and S. S. Kumar, "Simulation and comparison of spwm and svpwm control for three phase inverter," *ARPN journal of engineering and applied sciences*, vol. 5, no. 7, pp. 61–74, 2010.

[25] S. N. Vukosavic, *Grid-side converters control and design.* Springer, 2018.

[26] C. J. O'Rourke, M. M. Qasim, M. R. Overlin, and J. L. Kirtley, "A geometric interpretation of reference frames and transformations: dq0, clarke, and park," *IEEE Transactions on Energy Conversion*, vol. 34, no. 4, pp. 2070–2083, 2019.

[27] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, p. 1–122, Jan. 2011.

[28] M. R. Hestenes, "Multiplier and gradient methods," *Journal of optimization theory and applications*, vol. 4, no. 5, pp. 303–320, 1969.

[29] M. J. Powell, "A method for nonlinear constraints in minimization problems," *Optimization*, pp. 283–298, 1969.

[30] J. Rawlings, D. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017. [Online]. Available: https://books.google.cl/books?id=MrJctAEACAAJ

[31] T. L. Floyd and D. M. Buchla, *Electric circuits fundamentals*. Pearson/Prentice Hall, 2004.

[32] Introduction to vitis hls - vitis high level synthesis user guide (ug1399). [Online]. Available: https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/Introduction-to-Vitis-HLS

[33] Using directives in scripts vs. pragmas in code - vitis high level synthesis user guide (ug1399). [Online]. Available: https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/Using-Directives-in-Scripts-vs.-Pragmas-in-Code

[34] Floats and doubles - vitis high level synthesis user guide (ug1399). [Online]. Available: https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/Floats-and-Doubles

[35] config_compile - vitis high level synthesis user guide (ug1399). [Online]. Available: https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/config_compile

[36] Vivado design suite user guide: Design flows overview (ug892). [Online]. Available: https://docs.xilinx.com/r/en-US/ug892-vivado-design-flows-overview

[37] Pynq introduction - python productivity for zynq (pynq). [Online]. Available: https://pynq.readthedocs.io/en/latest/