



UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA

**ANALIZADOR MULTICANAL DE SEÑALES ELÉCTRICAS
BASADO EN SOC FPGA**

TESIS PROFESIONAL
QUE PARA OBTENER EL TÍTULO DE:
INGENIERO ELÉCTRICO-ELECTRÓNICO
P R E S E N T A:
JUAN JOSÉ LÓPEZ SOLÓRZANO

ASESOR:
DR. J. GUADALUPE VELÁSQUEZ AGUILAR

CUERNAVACA, MORELOS

FEBRERO, 2020

AGRADECIMIENTOS

Quiero agradecer...

A la Universidad Autónoma del Estado de Morelos, en especial a la Facultad de Ciencias Químicas e Ingeniería por los gratos recuerdos de mi carrera.

Al personal docente de la facultad, agradezco las enseñanzas transmitidas por cada uno de ustedes a lo largo de mi formación profesional.

A mi asesor el Dr. J. Guadalupe Velásquez Aguilar por darme la oportunidad del desarrollo de este trabajo, su asesoría, consejos y tiempo dedicado para aclarar mis dudas.

A todos mis compañeros con los que compartí un aula.

A mis amigos Christian y Mariano por su amistad incondicional.

A mi amigo de toda la vida David.

A mis amigos y compañeros de la facultad, Alejandro, Hoguer, José, Samuel, Eduardo, Bryan, Lorenzo, Ulises, Alondra, Alan y Kenia.

A mis tíos y tías, Jorge, Lupita, Cele, Letty, Mire, Miguel, Chin, Marisol, Mary y Laura.

A mis primos y primas Rocío, Jillos, Daniel, Raquel, Raúl, Lizzie, Luis Martín, Gallo, José Luis, Claudia, Carlos, Valeria, Cynthia, Alejandro, Leo, Fernando, José Antonio, Gerardo, Rebeca, Diana, María Ángel, Armando, Michel y Miguel.

DEDICATORIAS

Quiero dedicar este trabajo...

A DIOS porque él es todo y gracias a él tengo lo que tengo y soy lo que soy.

A mis padres Lorena y Juan José, que con su amor, cariño, enseñanzas y ejemplo he logrado todo lo que me he propuesto. Los quiero.

A mis abuelitos Raquel y Gerónimo, que dejaron este mundo, pero no sin antes dejar su legado en mí, no les fallaré...

A mis abuelitos Ma. Soledad y José Luz, ustedes son un gran ejemplo de amor para todos sus nietos.

RESUMEN

Este trabajo presenta el diseño e implementación de un sistema para obtener visualización de señales eléctricas, espectro de frecuencias y parámetros eléctricos utilizando la tecnología SoC FPGA incluida en la tarjeta de desarrollo de la marca Intel DE1 SoC. El sistema propuesto es capaz de adquirir señales de voltaje y corriente a una frecuencia de muestreo de hasta 124kHz simultáneamente. El algoritmo FFT se utiliza para obtener el espectro de frecuencia y calcular la distorsión armónica total (THD) de la señal eléctrica. Los resultados del sistema son desplegarlos en su propia interfaz gráfica de usuario (GUI) propia. La fiabilidad del sistema se valida comparando los resultados con los equipos de metrología eléctrica.

Palabras clave— SoC, FPGA, Sistemas empotrados, Calidad de la energía, Espectro de Frecuencias, Monitoreo Open-Source.

ABSTRACT

This work presents the design and implementation of a system to obtain visualization of electrical signals, Frequency Spectrum and electrical parameters using the SoC FPGA technology included in the Intel DE1 SoC development board. The proposed system is capable of acquiring voltage and current signals at a sampling rate of up to 124 kHz simultaneously. FFT algorithm is used to obtain the Frequency Spectrum and compute Total Harmonic Distortion (THD) of the electrical signal. System results are displayed in its own graphical user interface (GUI). The reliability of the system is validated by comparing the results with the electrical metrology equipment.

Keywords— SoC, FPGA, Embedded Systems, Power Quality, Frequency Spectrum, Open-Source Monitoring.

INDICE

Agradecimientos	I
Dedicatorias	II
Resumen	III
Abstract	IV
Indice	V
Indice de figuras	VIII
CAPÍTULO 1: INTRODUCCIÓN	1
1.1 Antecedentes	1
1.2 Motivación	2
1.3 Justificación	3
1.4 Objetivos	4
1.4.1 Objetivo general	4
1.4.2 Objetivos específicos	4
1.5 Alcances	4
1.6 Descripción del sistema	5
1.7 Organización de la tesis	7
CAPÍTULO 2: ADQUISICIÓN, PROCESAMIENTO Y ACONDICIONAMIENTO DE SEÑALES	8

2.1	Lenguajes descriptivos de hardware	8
2.1.1	Verilog	9
2.1.2	VHDL	9
2.2	Software de apoyo	10
2.2.1	Quartus	10
2.3	Controlador del convertidor analógico digital LTC 2303	11
2.3.1	Convertidores analógico-digitales	11
2.3.2	ADC empotrado en el SoC FPGA	12
2.3.3	Desarrollo de la arquitectura para conversión	16
2.4	Implementación de la transformada rápida de Fourier	26
2.4.1	Transformada rápida de Fourier	26
2.4.2	Memorias FIFO	31
2.4.3	Desarrollo de la arquitecturas para obtención de la FFT de la señal	33
2.5	Circuito de acondicionamiento de señales de entrada para el ADC	41
2.5.1	Acondicionamiento de señal	41
2.5.2	Amplificadores operacionales	42
2.5.3	Desarrollo del circuito de acondicionamiento	43
CAPÍTULO 3: ENLACE HPS-FPGA		66
3.1	Introducción	66
3.2	Tecnologías reconfigurables	66
3.2.1	CPLD	66
3.2.2	FPGA	67
3.3	Procesadores empotrados (ARM)	69
3.4	Sistemas en un chip (SoC)	70
3.5	SoC-FPGA	70
3.5.1	HPS-FPGA Bridges	72
3.5.2	Linux para sistemas empotrados	74
3.6	Enlace entre FPGA y HPS con Linux Ubuntu gráfico	77
CAPÍTULO 4: INTERFAZ GRÁFICA DE USUARIO		90
4.1	Introducción	90
4.2	Lenguaje C	90
4.2.1	Punteros en C	91
4.3	Java	92
4.3.1	Java AWT Y Swing	93
4.3.2	JNI	93
4.3.3	eclipse	95

4.4	Controlar y leer señales de un SoC FPGA mediante una interfaz gráfica de usuario	96
CAPÍTULO 5: PRUEBAS Y RESULTADOS		128
5.1	Prueba de controlador del ADC con potenciómetro	128
5.2	Prueba de FFT y FIFO	130
5.3	Prueba en simulador del circuito de acondicionamiento de señales	135
5.4	Sistema en gabinete	139
5.5	Prueba de enlace HPS-FPGA	141
5.6	Pruebas de interfaz gráfica de usuario	143
5.7	Resultados	146
5.8	Conclusiones y recomendaciones	150
5.8.1	Conclusiones	151
5.8.2	Recomendaciones	152
Apéndices		154
A.	Instalación de Quartus	154
B.	Diseño de PCB en Proteus	158
C.	Montar Linux Ubuntu en microSD	161
D.	Instalación de eclipse	164
E.	Instalación de Cygwin	165
F.	Instalación de CDT (C/C++ Development Tool)	173
G.	Agregar librerías a eclipse	175
H.	Descarga de SoC EDS (Embedded Development Suite)	177
I.	Instalación de Java y gcc en Linux Ubuntu	177
Bibliografía		182

INDICE DE FIGURAS

CAPÍTULO 1: INTRODUCCIÓN

Figura 1.1 Diagrama de bloques del sistema	6
--	---

CAPÍTULO 2: ADQUISICIÓN, PROCESAMIENTO Y ACONDICIONAMIENTO DE SEÑALES

Figura 2.1 Señales del Header de 2x5, [5].	13
Figura 2.2 Conexiones entre el FPGA, el header de 2x5 y el ADC, [5].	13
Figura 2.3 Palabra de datos de entrada, [6].	14
Figura 2.4 Configuración del canal, [6].	14
Figura 2.5 Ejemplos de la configuración del MUX, [6].	15
Figura 2.6 Temporización del LTC2308 con un pulso CONVST largo, [6].	16
Figura 2.7 Temporización del LTC2308 con un pulso CONVST corto, [6].	16
Figura 2.8 Entidad del código ADV_ADC.	17
Figura 2.9 Simulación del código ADV_ADC.	20
Figura 2.10 Entidad del programa ADC_MEGA.	21
Figura 2.11 Diagrama de estados de la FSM utilizada en el código ADC_MEGA.	21
Figura 2.12 Diagrama de bloques del código ADC_module.	25
Figura 2.13 Comparación del funcionamiento de un registro convencional y uno FIFO, [9].	32
Figura 2.14 Diagrama de bloques de una memoria FIFO serie típica, [9].	32
Figura 2.15 Ejemplos de registro FIFO para aplicaciones de control de la velocidad de transmisión de datos, [9].	33
Figura 2.16 Modelo de Simulink.	34
Figura 2.17 Entidad del programa generado con HDL Coder.	34
Figura 2.18 Simulación del código generado con HDL Coder,	35
Figura 2.19 Señal de entrada y su espectro de frecuencias.	36
Figura 2.20 Comparación de la salida del código en VHDL contra los resultados obtenidos utilizando la herramienta Matlab.	36
Figura 2.21 Entidad del programa ADC_FFT.	37
Figura 2.22 ADC_module.	38
Figura 2.23 Memoria FIFO asíncrona.	38
Figura 2.24 Simulación de la memoria FIFO.	39
Figura 2.25 Entidad de ADC_FIFO.	40
Figura 2.26 Arquitectura del sistema.	41
Figura 2.27 Pinout de un Amplificador Operacional.	42
Figura 2.28 Circuito genérico propuesto para cada canal.	44

Figura 2.29 Circuito de acondicionamiento de señales para cuatro canales.	45
Figura 2.30 Circuito genérico para cada canal sin transformador.	46
Figura 2.31 Primera etapa del circuito.	46
Figura 2.32 Circuito equivalente de la primera etapa del circuito.	47
Figura 2.33 Segunda etapa del circuito.	50
Figura 2.34 Circuito equivalente de la segunda etapa del circuito.	50
Figura 2.35 Comportamiento del circuito cuando se mide voltaje fase A, B o C en 127 V.	53
Figura 2.36 Comportamiento de la amplificación en el circuito para voltaje fases A, B o C a 127 V en base a los potenciómetros.	54
Figura 2.37 Comportamiento del offset en el circuito para voltaje fases A, B o C a 127 V en base a los potenciómetros.	55
Figura 2.38 Comportamiento del circuito cuando se mide voltaje fase A, B o C en 220 V.	56
Figura 2.39 Comportamiento de la amplificación en el circuito para voltaje fases A, B o C a 220 V en base a los potenciómetros.	57
Figura 2.40 Comportamiento del offset en el circuito para voltaje fases A, B o C a 220 V en base a los potenciómetros.	58
Figura 2.41 Comportamiento del circuito cuando se mide corriente en 127 V.	59
Figura 2.42 Comportamiento de la amplificación en el circuito para corriente en 127 V en base a los potenciómetros.	61
Figura 2.43 Comportamiento del offset en el circuito para corriente en 127 V en base a los potenciómetros.	61
Figura 2.44 Comportamiento del circuito cuando se mide corriente en 220 V.	63
Figura 2.45 Comportamiento de la amplificación en el circuito corriente a 220 V en base a los potenciómetros.	64
Figura 2.46 Comportamiento del offset en el circuito para corriente a 220 V en base a los potenciómetros.	64

CAPÍTULO 3: ENLACE HPS-FPGA

Figura 3.1 Modelo de FPGA SoC de Altera (Intel).	71
Figura 3.2 Conjuntos de SoC FPGA de Altera (Intel).	71
Figura 3.3 Procesadores ARM.	72
Figura 3.4 Diagrama de bloques de un dispositivo SoC de la marca Altera (Intel), [18].	72
Figura 3.5 Conectividad de los HPS-FPGA Bridges, [18].	73
Figura 3.6 Dirección base para acceder al Lightweight HPS-FPGA bridge, [18].	74
Figura 3.7 Componente de HPS en Qsys.	74
Figura 3.8 Linux Yocto en el SoC FPGA.	76
Figura 3.9 Linux Ubuntu en SoC FPGA.	76
Figura 3.10 Proyecto descargado DE1_SOC_Linux_Audio.	77
Figura 3.11 Archivos del sistema de adquisición de señales añadidos.	78
Figura 3.12 Archivo descargado: soc_system.qsys.	78
Figura 3.13 Ventana de configuración de PIO (Componente de Qsys).	80
Figura 3.14 Creación de nuevo componente en Qsys.	81
Figura 3.15 Adición de los archivos VHDL al componente creado para Qsys.	81
Figura 3.16 Asignación de señales para el componente creado para Qsys.	82

Figura 3.17 Configuración de interfaces para el componente creado para Qsys.	83
Figura 3.18 Componente creado para Qsys.	83
Figura 3.19 Qsys generado.	84
Figura 3.20 Archivo de verilog a eliminar.	85
Figura 3.21 Generación de archivos VHDL del sistema realizado en Qsys.	86
Figura 3.22 Entidad del archivo generado.	86
Figura 3.23 Máxima entidad del sistema analizador multicanal.	87
Figura 3.24 Convertidor de archivos de salida de Quartus.	88
Figura 3.25 Archivo “soc_system.rbf” generado en Quartus montado en la microSD.	88
Figura 3.26 Configuración del selector MSEL de la tarjeta DE1 SoC, [22].	89
Figura 3.27 Conexiones y equipo requerido para utilizar el FPGA en modo gráfico.	89

CAPÍTULO 4: INTERFAZ GRÁFICA DE USUARIO

Figura 4.1 Rol del JNI en una aplicación, [29].	94
Figura 4.2 BorderLayout de Java Swing.	96
Figura 4.3 FlowLayout de Java Swing.	97
Figura 4.4 FlowLayout aplicado en la parte “NORTH” del BorderLayout.	97
Figura 4.5 Ejemplo de GridLayout de 2x3.	98
Figura 4.6 GridLayout aplicado a la parte “CENTER” de un BorderLayout.	98
Figura 4.7 Interfaz gráfica programada.	99
Figura 4.8 Cruces por ceros que pueden ser detectados con el algoritmo.	102
Figura 4.9 Cruces por cero detectados y cruce por cero aproximado por la FFT.	103
Figura 4.10 Primer paso para convertir un proyecto a un proyecto de C/C++.	106
Figura 4.11 Convertir un proyecto a proyecto de C/C++.	107
Figura 4.12 Configuración del proyecto de C.	107
Figura 4.13 Carpeta donde se almacenarán los programas de C.	108
Figura 4.14 Crear un target.	108
Figura 4.15 Configuración del nuevo target.	109
Figura 4.16 Habilitación del comando Javah.	109
Figura 4.17 Ubicación de javah.exe.	110
Figura 4.18 Selección del Workspace Directory.	111
Figura 4.19 Configuración de la herramienta externa Javah.	111
Figura 4.20 Instrucción de construir un target.	112
Figura 4.21 Creación de un programa en C.	112
Figura 4.22 Configuración inicial del programa en C.	113
Figura 4.23 Ubicación de hwlib.h en la carpeta de instalación del SoC EDS.	114
Figura 4.24 Ubicación de alt_gpio.h en la carpeta de instalación del SoC EDS.	114
Figura 4.25 Ubicación de hps.h y socal.h en la carpeta de instalación del SoC EDS.	115
Figura 4.26 Ubicación del SoC Embedded Command Shell.	115
Figura 4.27 Embedded Command Shell.	116
Figura 4.28 Generación del archivo de cabecera hps_0.h.	117
Figura 4.29 Librería hps_0.h generada a partir del Qsys del proyecto de Quartus para FPGA.	117
Figura 4.30 Creación del target “all”.	119
Figura 4.31 Correr Javah de nuevo.	120

Figura 4.32	Generando “all”.	120
Figura 4.33	Archivos del proyecto de eclipse necesarios.	121
Figura 4.34	Librerías .jar que se instalarán en Linux.	121
Figura 4.35	Archivos en Linux.	122
Figura 4.36	Librerías en Linux.	122
Figura 4.37	Terminal en la sesión del root.	123
Figura 4.38	Carpeta con archivos trasladados.	123
Figura 4.39	Inclusión de librería JFreeChart en Linux.	124
Figura 4.40	Compilación de los archivos de Java por medio del comando javac.	125
Figura 4.41	Creación del archivo de cabecera JNIserver.h por medio del comando javah.	125
Figura 4.42	Compilación de programa en C para creación de librerías compartidas para acceder a las funciones JNI del programa desde Linux.	126
Figura 4.43	Correr la interfaz gráfica desde terminal en FPGA SoC.	127
Figura 4.44	Interfaz gráfica en Linux Ubuntu de la DE1 SoC.	127

CAPÍTULO 5: PRUEBAS Y RESULTADOS

Figura 5.1	Prueba con potenciómetro.	129
Figura 5.2	Frecuencia de muestreo observada en el osciloscopio.	130
Figura 5.3	Señal de prueba.	133
Figura 5.4	Sistema de pruebas.	133
Figura 5.5	Datos recibidos en la terminal serial.	134
Figura 5.6	Datos obtenidos graficados en Excel.	134
Figura 5.7	Datos obtenidos y su FFT en Matlab.	134
Figura 5.8	Comparación de datos de la FFT obtenida con el sistema contra los datos obtenidos por el algoritmo de Matlab.	135
Figura 5.9	Circuito de pruebas.	135
Figura 5.10	Prueba del circuito para medición de voltajes fase A, B o C a 127V.	136
Figura 5.11	Prueba del circuito para medición de voltajes fase A, B o C a 220V.	137
Figura 5.12	Prueba del circuito para medición de corrientes a 127 V.	138
Figura 5.13	Prueba del circuito para medición de corrientes a 220 V.	138
Figura 5.14	Circuito implementado en el gabinete junto con el FPGA y los transformadores de medición.	139
Figura 5.15	Vista superior del gabinete.	140
Figura 5.16	Vista trasera del gabinete.	140
Figura 5.17	Vista frontal del gabinete.	141
Figura 5.18	Linux Ubuntu funcionando con el FPGA.	142
Figura 5.19	Escritorio de Linux Ubuntu para DE1 SoC.	142
Figura 5.20	Prueba de programación en Java 1.	144
Figura 5.21	Prueba de programación en Java 2.	145
Figura 5.22	Prueba de programación en Java 3.	145
Figura 5.23	Prueba de programación en Java 4.	145
Figura 5.24	Equipo de prueba Dranetz Power Visa	147
Figura 5.25	Centro de carga del TAMULBA	147
Figura 5.26	Motor de bomba de agua	148

Figura 5.27 Sistema instalado para las pruebas	148
Figura 5.28 Conexiones para la prueba	149
Figura 5.29 Resultados obtenidos con el sistema	150

CAPÍTULO 1: INTRODUCCIÓN

1.1 ANTECEDENTES

La calidad de la energía es un aspecto fundamental en la industria eléctrica, esta se define como la ausencia de interrupciones, sobretensiones y deformaciones producidas por fallas o por la operación de equipos con tecnología basada en semiconductores que provocan armónicos en la red eléctrica, tanto en servicio doméstico, el comercial y en el industrial, así como en la red del suministrador. El monitoreo continuo de la calidad de la energía es necesario para obtener información sobre problemas asociados con la operación de una red eléctrica, los cuales generalmente están relacionados con:

1. Detectar desgaste del embobinado de transformadores debido a sobrecargas de corriente.
2. Impedir penalizaciones en el cobro por bajo factor de potencia.
3. Reconocer variaciones en la frecuencia del sistema eléctrico.
4. Prevenir daños en el sistema eléctrico por variaciones de tensión.

En 2016 se creó en México el Código de Red de la Comisión Reguladora de Energía, el cuál es un documento donde se encuentran establecidas las disposiciones administrativas de carácter general que establecen los criterios para la regulación de la

calidad de energía respecto a eficiencia, confiabilidad, continuidad, seguridad y sustentabilidad del Sistema Eléctrico Nacional.

Este documento también establece multas económicas indicadas en sus apartados derivadas de la no atención del monitoreo de la energía en centros de carga en Media o Alta Tensión, que van del 2% al 10% de los ingresos brutos del usuario. Estas reglas de operación entraron en vigor a partir del 2019 y de aquí la importancia de equipos de monitoreo.

Por otro lado, en aspectos tecnológicos, en la actualidad el desarrollo de la tecnología digital ha crecido exponencialmente. Las diversas variedades disponibles como lo son: TTL (Transistor-Transistor Logic), CMOS (Complementary Metal Oxide Semiconductor); lógica programable: CPLD (Complex Programmable Logic Device), FPGA (Field Programmable Gate Array), ASIC (Application Specific Integrated Circuits), microcontroladores, microprocesadores, memorias RAM, entre otros, son utilizadas en el desarrollo de circuitos y sistemas digitales.

El diseño y desarrollo de sistemas, generalmente requiere de varias de estas tecnologías estén trabajando de forma conjunta para poder realizar tareas complejas. En muchas ocasiones el sistema llega a ser tan complejo que, la coordinación de las diversas tecnologías resulta una extremadamente difícil de lograr, de ahí que una alternativa para estos casos es el uso de los SoC (System on a Chip).

Un SoC engloba varias tecnologías en un solo chip, reduciendo así el trabajo de coordinación entre los elementos que lo componen y evitando realizar complicados métodos de comunicación entre los componentes.

1.2 MOTIVACIÓN

El monitoreo de señales eléctricas se puede lograr con un analizador de la calidad de la energía (PQA, Power Quality Analyzer). Uno de los inconvenientes de estos equipos es su costo elevado, por lo que una alternativa es utilizar tecnología de bajo costo para

realizar esta tarea y la propuesta de este trabajo es utilizar tarjetas basadas en SoC FPGA cuyo código está disponible para diversas aplicaciones y de fácil acceso.

La tecnología SoC FPGA aún no es una tecnología muy utilizada por su reciente incorporación, por lo que en este trabajo, se establecerá la metodología de cómo realizar un sistema completo basado en esta tecnología, utilizando recursos de bajo costo y con software con licencia GNU GPL (General Public License). Demostrando así como el SoC FPGA es una alternativa viable para el desarrollo de sistemas de monitoreo y procesamiento basados en microprocesadores.

1.3 JUSTIFICACIÓN

En el área de los sistemas de potencia, una limitante de los sistemas de monitoreo de la calidad de la energía, es el costo de los equipo, ya que los precios son bastante elevados y con las reformas actuales respecto a la calidad de energía, actualmente es necesario contar con uno.

En este trabajo se busca la forma de crear un sistema basado en un SoC FPGA que englobe en un solo chip, un FPGA que por sus propiedades pueda permitir desarrollar alguna arquitectura y un microprocesador que sea capaz de soportar un sistema operativo gráfico que a su vez coordine las entradas y salidas de un FPGA.

Por lo anterior, la propuesta que se presenta en este trabajo tendrá dentro de una de sus ventajas que es de bajo costo, comparado con los dispositivos comerciales y que adicionalmente, tenga una interfaz gráfica que permitirá al usuario interactuar para el cambio de parámetros de adquisición y despliegue.

Al no contar con certificaciones el equipo puede funcionar como equipo de monitoreo personal, para saber que todo se encuentra en orden y al momento de una auditoria con un equipo certificado no exista problema alguno.

1.4 OBJETIVOS

1.4.1 OBJETIVO GENERAL

Realizar un sistema de aplicación específica, basado en un SoC FPGA que permita realizar la adquisición, almacenamiento (vía Hardware) y despliegue de señales eléctricas, para monitorear en tiempo-real los parámetros básicos que debe realizar un sistema de monitoreo de la calidad de la energía (voltaje eficaz, corriente eficaz, frecuencia, factor de potencia y distorsión armónica total (THD)).

1.4.2 OBJETIVOS ESPECÍFICOS

1. Desarrollo de las arquitecturas hardware para la adquisición y almacenamiento de la señal.
2. Compilación del sistema operativo acorde a la arquitectura hardware diseñada.
3. Configuración para enlazar software-hardware del sistema.
4. Banco de pruebas para la validación de la funcionalidad del sistema.

1.5 ALCANCES

Este trabajo presentará un sistema que permite inspeccionar parámetros eléctricos de un dispositivo o de una instalación conectada a la red eléctrica (monofásica o trifásica). La adquisición de las señales se realiza de forma no intrusiva. Derivado de que los datos pueden ser monitoreados en tiempo real, el sistema puede ser utilizado para adquisición a alta velocidad de señales multiplexando hasta 8 canales del ADC. El uso de sistema operativo para controlar el procesador empujado, facilita el despliegue de la información. Finalmente, la tecnología de bajo costo y de código abierto, permite que esté al alcance de todos, ya que con un bajo recurso de inversión, se puede obtener un dispositivo de adquisición y procesamiento de señales flexible y con la calidad de los equipos comerciales.

1.6 DESCRIPCIÓN DEL SISTEMA

El sistema a realizarse es un analizador de la calidad de la energía y parámetros eléctricos, el cual tomará la lectura de 4 diferentes señales (Las tres fases de voltaje y la corriente de la fase A) y calculará su distorsión armónica total (THD) por medio del espectro de frecuencias, valores eficaces, frecuencia de la señal, y el factor de potencia de la fase A así como sus potencias real, reactiva y aparente.

Las señales serán obtenidas de la red eléctrica y leídas por un ADC LTC2308 incorporado en la tarjeta DE1 SoC. Para que las señales puedan ser leídas por el ADC se diseñó un circuito de acondicionamiento de señales el cual reduce la tensión a niveles que puedan ser leídos por el ADC.

Una vez leídas por el ADC se utilizará una arquitectura FPGA que sea capaz de obtener digitalizadas las señales convertidas por el ADC. Dentro de dicha arquitectura se utilizará también el algoritmo de la Transformada Rápida de Fourier para obtener el espectro de frecuencias de las señales adquiridas. Y para no perder ningún dato se utilizará una memoria circular FIFO. La arquitectura debe ser capaz también de controlar la frecuencia de muestreo y los relojes de escritura y lectura de la memoria FIFO.

Por medio del enlace entre FPGA y procesador empotrado que permite la tecnología FPGA SoC, se enlazarán las señales de entradas y salidas de la arquitectura FPGA (Hardware) a las localidades de memoria del procesador (Software) con el fin de poder controlarlas por medio de este.

El procesador es capaz de soportar un sistema operativo Linux Ubuntu de manera gráfica, en el cual se utilizará una interfaz gráfica de usuario encargada de controlar y leer las señales de la arquitectura FPGA, además de eso será la encargada del despliegue de las señales discretas adquiridas pudiendo seleccionar el número de muestras a adquirir y la frecuencia de muestreo, también desplegará el espectro de frecuencias de cualquiera de las 4 señales adquiridas y calculará los parámetros eléctricos descritos anteriormente.

El diagrama de bloques del sistema se puede observar en la Figura 1.1

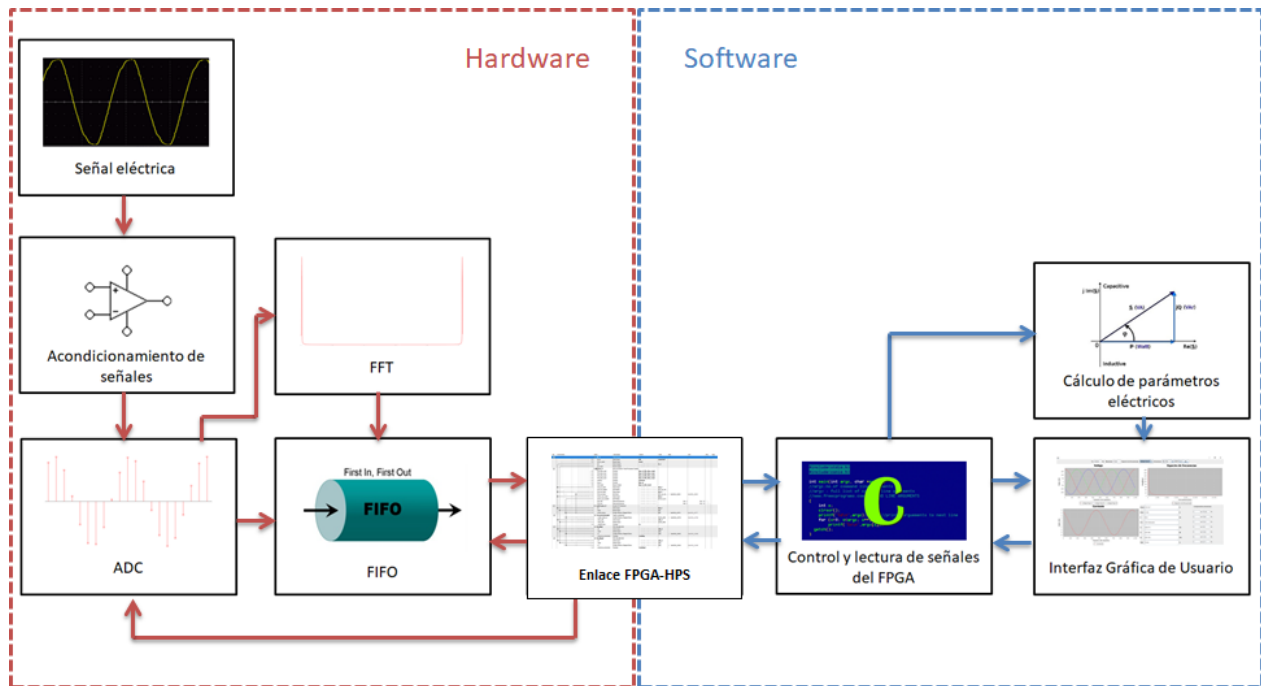


Figura 1.1 Diagrama de bloques del sistema

La Figura 1.1 se desglosa de la siguiente manera:

- Señal eléctrica: Puede ser una señal de voltaje a 127 V o a 220 V, o una señal de corriente que provenga de un sistema a 127 V o a 220 V.
- Acondicionamiento de señales: Circuitería analógica que permite que la señal eléctrica pueda ser leída por el convertidor analógico digital (ADC).
- ADC: Implica tanto el ADC como hardware así como el controlador diseñado en el FPGA encargado de adquirir el dato convertido por el ADC, controlar la frecuencia de muestreo y coordinación del canal donde se adquiera la señal (multiplicación de canales).
- FFT: Transformada Rápida de Fourier de las muestras adquiridas por el ADC, se obtiene por medio del diseño de una arquitectura para FPGA.
- FIFO: Funciona para no perder ningún dato que adquiera el ADC, además almacena los resultados de la FFT.
- Enlace FPGA-HPS: En este punto se enlaza el FPGA con el procesador, para que las entradas y salidas del FPGA pueden ser controladas o leídas, según sea el caso, desde el procesador.

- Control y lectura de señales del FPGA: Por medio de programación básica de software en el procesador las señales del FPGA pueden ser leídas como localidades de memoria de lectura o controladas como localidades de memoria de escritura.
- Cálculo de parámetros eléctricos: Con las señales adquiridas es posible calcular los parámetros eléctricos que se deben monitorear para la calidad de la energía como: voltaje eficaz, corriente eficaz, frecuencia, factor de potencia, potencia real, aparente y reactiva, THDV, THDI y las componentes armónicas.
- Interfaz gráfica de usuario: Es una ventana con la que el usuario puede interactuar y se encarga de coordinar el software (Control y lectura de señales del FPGA y Cálculo de parámetros eléctricos).

1.7 ORGANIZACIÓN DE LA TESIS

El presente trabajo consta de cinco capítulos. En el capítulo dos se describen los procesos y arquitecturas de los módulos de adquisición, procesamiento y acondicionamiento de señales, que fueron requeridos para realizar la tarea deseada.

En el tercer capítulo se hace una revisión de la tecnología a utilizar (FPGA, Procesadores empotrados y SoC FPGA) y se muestra como realizar el enlace entre hardware (FPGA) y el procesador empotrado (Hard Processor System, HPS).

En el cuarto capítulo se describe el desarrollo en software de la interfaz gráfica que interactúa con los enlaces entre el FPGA y el microprocesador a los cuales están conectadas las entradas y salidas de una arquitectura FPGA que se quieran controlar o leer.

En el quinto capítulo se presentan las pruebas y resultados del sistema realizado. Además se incluyen al final del mismo las conclusiones del trabajo.

Finalmente en la sección de apéndice, se incluyen las guías útiles para el desarrollo del sistema (instalación de software, creación de PCB, etc.)

CAPÍTULO 2: ADQUISICIÓN, PROCESAMIENTO Y ACONDICIONAMIENTO DE SEÑALES

2.1 LENGUAJES DESCRIPTIVOS DE HARDWARE

Un lenguaje de descripción de hardware (HDL), es un lenguaje de programación de software utilizado para modelar el funcionamiento previsto de una pieza de hardware. Hay dos aspectos en la descripción del hardware que facilita un HDL; modelado de comportamiento abstracto verdadero y modelado de estructura de hardware. [1]

Modelado de comportamiento abstracto: Un lenguaje de descripción de hardware es declarativo para facilitar la descripción abstracta del comportamiento del hardware para fines de especificación. Este comportamiento no se ve perjudicado por los aspectos estructurales o de diseño de la intención del hardware.

Modelado de estructura de hardware: La estructura de hardware puede modelarse en un lenguaje de descripción de hardware, independientemente del comportamiento del diseño.

El comportamiento del hardware puede ser modelado y representado en varios niveles de abstracción durante el proceso de diseño. Los modelos de nivel superior describen el funcionamiento del hardware de manera abstracta, mientras que los modelos de nivel inferior incluyen más detalles, como la estructura de hardware inferida

2.1.1 VERILOG

Verilog HDL es un lenguaje de descripción de hardware utilizado para diseñar y documentar sistemas electrónicos. Verilog HDL permite a los diseñadores diseñar en varios niveles de abstracción. Es el HDL más utilizado con una comunidad de usuarios de más de 50,000 diseñadores activos. [2]

2.1.2 VHDL

A partir del desarrollo de circuitos integrados digitales programables con una gran cantidad de componentes lógicos y la necesidad de sistemas digitales para aplicaciones más complejas, las herramientas de diseño tradicionales se vuelven cada vez más ineficientes e ineficaces para lograr desarrollos adecuados, por lo tanto, las empresas fabricantes de circuitos integrados se vuelven más útiles herramientas, originando así los lenguajes de descripción de hardware o HDL. Por lo tanto, cada empresa crea la suya estableciendo una gran diversidad de idiomas. [1]

Para tratar de unificar estas herramientas, entre 1984 y 1987, el IEEE y el Departamento de Defensa de los Estados Unidos (DoD) patrocinan el desarrollo de un lenguaje llamado VHDL. Su nombre proviene de VHSIC HDL, es decir, lenguaje de descripción de hardware para circuitos integrados de muy alta velocidad.

Considerando que un lenguaje de descripción de hardware es una herramienta formal que permite describir la estructura y el comportamiento de un sistema para lograr una especificación, documentación y simulación adecuadas antes de su realización real;

para su implementación se establecieron ciertas características fundamentales, que aún son válidas hoy en día, y son:

- Cada elemento de diseño tiene una interfaz única y perfectamente definida, que le permite conectarse a otros elementos.
- Cada elemento tiene un comportamiento preciso y único, que permite su simulación posterior.
- La especificación de comportamiento que permite definir la operabilidad se puede hacer a través de un algoritmo o una estructura de hardware real.
- Los diseños mantienen una estructura jerárquica que le permite descomponerse adecuadamente.
- Las características concurrentes, temporizadas y de sincronización (p. Ej. Reloj) se pueden modelar correctamente.
- Se puede simular cualquier operación de lógica y temporización.

Esto establece una herramienta que también tiene amplias características de modelado y documentación. De esta forma, cualquier circuito digital puede especificarse y simularse correctamente.

Después del desarrollo de este lenguaje, aparecieron herramientas de síntesis adecuadas que completan el panorama de diseño de un sistema digital. Por lo tanto, se puede decir que si usa VHDL puede diseñar, simular y sintetizar cualquier sistema digital, desde la estructura secuencial combinatoria más simple hasta la más compleja. Las nuevas versiones de HDL también permiten el desarrollo de circuitos analógicos.

2.2 SOFTWARE DE APOYO

2.2.1 QUARTUS

El software Quartus II organiza y gestiona los elementos de un diseño dentro de un proyecto. El proyecto encapsula información sobre su jerarquía de diseño, bibliotecas, restricciones y configuraciones del proyecto. Quartus es el software de Altera mediante

el cual entre muchas otras cosas pueden hacerse proyectos basados en VHDL o manejo de sistemas con herramientas como Qsys [3].

Al abrir un proyecto, una GUI unificada muestra información integrada del proyecto. El navegador de proyectos le permite ver y editar los elementos de su proyecto. La ventana Mensajes enumera información importante sobre el procesamiento del proyecto.

Se puede guardar varias revisiones de un proyecto para experimentar con configuraciones que logren sus objetivos de diseño. Los proyectos Quartus II admiten flujos de trabajo distribuidos y basados en equipo y una interfaz de secuencias de comandos.

El proceso de instalación del software Quartus se puede revisar en el Apéndice A.

2.3 CONTROLADOR DEL CONVERTIDOR ANALÓGICO DIGITAL LTC 2303

2.3.1 CONVERTIDORES ANALÓGICO-DIGITALES

Los dispositivos ADC convierten un nivel de tensión analógico en una palabra digital correspondiente. Si “n” es el número de bit obtenidos de la palabra, esto significa que habrá 2^n niveles de tensión diferentes. Todo convertidor ADC debe procurar que el conjunto de bit obtenidos a la salida sea un reflejo lo más exacto posible del valor analógico correspondiente. Se usan un gran número de métodos para convertir señales analógicas a la forma digital, los que más usados son: rampa de escalera, aproximaciones sucesivas, paralelo (flash), doble rampa, voltaje a frecuencia, tipo serie [4].

Dentro de los nuevos dispositivos, están los de conversión serie, la cual permite entregar una secuencia digital de ocho bit (o más) de salida en forma serial. La transmisión serie emplea una sola trayectoria para transportar bits de información, lo que la hace ideal para comunicaciones a grandes distancias, por su bajo costo en

cableado. Esta transmisión es realizada de forma síncrona o asíncrona. Muchos de estos dispositivos están basados en el método de capacitor conmutado, el cual se describe a continuación.

La data paralelo entra al conversor de capacitor conmutado, que corresponde a una red de condensadores en serie a cada bit y a un interruptor conectado al voltaje de referencia (V_{ref}). Este valor de voltaje establece los límites superiores e inferiores de la salida analógica. La conversión está directamente relacionada con el valor de tensión que carga el condensador, cada condensador de la rama está ajustado según el bit que tenga a la entrada, lo que va a determinar el tiempo de carga. En la generación de la señal analógica, la carga de cada condensador es conmutada y sumada, obteniéndose la señal por intervalos de tiempo y suma en el punto inversor del amplificador operacional de la entrada.

2.3.2 ADC EMPOTRADO EN EL SOC FPGA

La tarjeta de desarrollo utilizada para el proyecto, la DE1-SoC, tiene un convertidor analógico-digital (LTC2308), tiene una relación señal/ruido (SNR) de 73.3dB, y ocho canales de 12 bits. Este ADC ofrece una tasa de rendimiento de conversión de hasta 500,000 muestras por segundo (sps, samples by second). El rango de entrada analógica para todos los canales de entrada puede ser de 0 V a 4.096V. El reloj de conversión interno permite que el reloj de datos de salida serie externa (SCLK) funcione a cualquier frecuencia de hasta 40MHz. Se puede configurar para aceptar ocho señales de entrada en las entradas ADC_IN0 a ADC_IN7. Estas ocho señales de entrada están conectadas a un header (conector) de 2x5 [5] como el que se observa en la Figura 2.1. Las conexiones entre la tarjeta de desarrollo DE1 SoC y el convertidor analógico digital LTC2308 se muestran en la Figura 2.2.

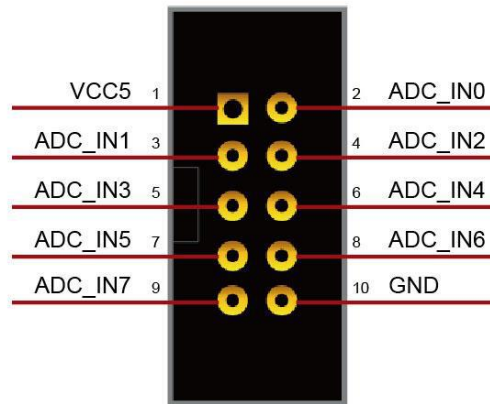


Figura 2.1 Señales del Header de 2x5, [5].

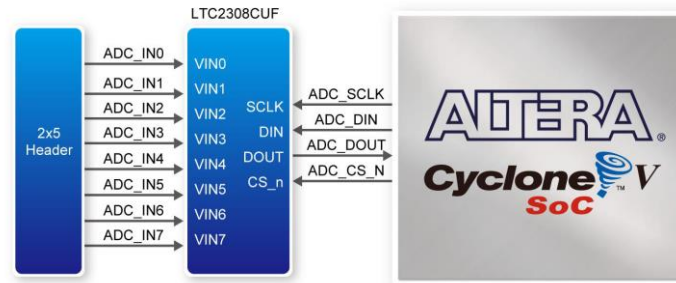


Figura 2.2 Conexiones entre el FPGA, el header de 2x5 y el ADC, [5].

El LTC2308 cuenta con una interfaz serial compatible con SPI / MICROWIRE. Este ADC incluye una referencia interna y un circuito de muestreo y retención completamente diferente para reducir el ruido en modo común. El LTC2308 opera desde un solo suministro de 5V y consume solo 3.5mA a una frecuencia de muestreo de 500ksps. La función de apagado automático reduce la corriente de suministro a 200µA a una frecuencia de muestreo de 1ksps. El LTC2308 está empaquetado en un pequeño QFN de 24 pines de 4mmx4mm. La referencia interna de 2.5V y el multiplexor de 8 canales reducen aún más los requisitos de espacio en la placa PCB. El bajo consumo de energía y el tamaño pequeño hacen que el LTC2308 sea ideal para aplicaciones operadas con batería y portátiles, mientras que la interfaz serial compatible con SPI de 4 hilos hace que este ADC sea una buena combinación para sistemas de adquisición de datos aislados o remotos, [6].

Los diversos modos de funcionamiento del LTC2308 se programan mediante una palabra DIN de 6 bits donde cada uno de los bits se puede observar en la Figura 2.3. Los bits de datos SDI se cargan en el borde ascendente de SCK, con el bit S/D cargado en el primer borde ascendente y el bit SLP en el sexto borde ascendente.

S/D	O/S	S1	S0	UNI	SLP
-----	-----	----	----	-----	-----

S/D = SINGLE-ENDED/DIFFERENTIAL BIT
O/S = ODD/SIGN BIT
S1 = ADDRESS SELECT BIT 1
S0 = ADDRESS SELECT BIT 0
UNI = UNIPOLAR/BIPOLAR BIT
SLP = SLEEP MODE BIT

Figura 2.3 Palabra de datos de entrada, [6].

La entrada analógica MUX está programada por los bits S/D, O/S, S1 y S0 de la palabra DIN. La Figura 2.4 enumera las configuraciones de MUX para todas las combinaciones de los bits de configuración. La Figura 2.5 muestra varias configuraciones posibles de MUX.

S/D	O/S	S1	S0	0	1	2	3	4	5	6	7	COM
0	0	0	0	+	-							
0	0	0	1			+	-					
0	0	1	0					+	-			
0	0	1	1							+	-	
0	1	0	0	-	+							
0	1	0	1			-	+					
0	1	1	0					-	+			
0	1	1	1							-	+	
1	0	0	0	+								-
1	0	0	1			+						-
1	0	1	0					+				-
1	0	1	1							+		-
1	1	0	0		+							-
1	1	0	1				+					-
1	1	1	0						+			-
1	1	1	1								+	-

Figura 2.4 Configuración del canal, [6].

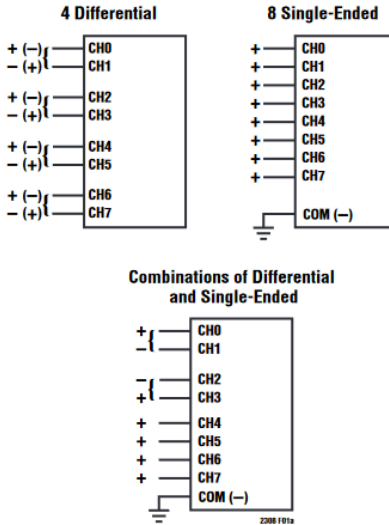


Figura 2.5 Ejemplos de la configuración del MUX, [6].

El inicio de una conversión se desencadena por un flanco ascendente en CONVST. Una vez iniciada, una nueva conversión no puede reiniciarse hasta que se complete la conversión actual. Las Figuras 2.6 y 2.7 muestran los diagramas de temporización para dos ejemplos diferentes de pulsos CONVST. El ejemplo 1 (Figura 2.6) muestra que CONVST permanece ALTO después de que finaliza la conversión. Si CONVST es alto después del período t_{CONV} , el LTC2308 ingresa al modo NAP o SLEEP, dependiendo de la configuración del bit SLP de la palabra DIN que se cambió después de la conversión anterior.

Cuando CONVST retoma el nivel bajo, el ADC se activa y el bit más significativo (MSB) de la secuencia de datos de salida en SDO se hace válido después de habilitar el bus de datos en serie. Todos los demás bits de datos de la transición SDO en el flanco descendente de cada impulso SCK. Los datos de configuración (DIN) se cargan en el LTC2308 en SDI, comenzando con el primer flanco ascendente SCK después de que CONVST regrese bajo. El bit S/D se carga en el primer flanco ascendente SCK.

El ejemplo 2 (Figura 2.7) muestra que CONVST regresa bajo antes de que finalice la conversión. En este modo, el ADC y todos los circuitos internos permanecen encendidos. Cuando se completa la conversión, el MSB de la secuencia de datos de salida en SDO se vuelve válido después de habilitar el bus de datos. En este punto

(t_{CONV} 1.3 μ s después del flanco ascendente de CONVST), SCK pulsante desplazará los datos en SDO y cargará los datos de configuración (DIN) en el LTC2308 en SDI. El primer flanco ascendente SCK carga el bit S/D en el LTC2308. Las transiciones SDO en el borde descendente de cada pulso SCK.

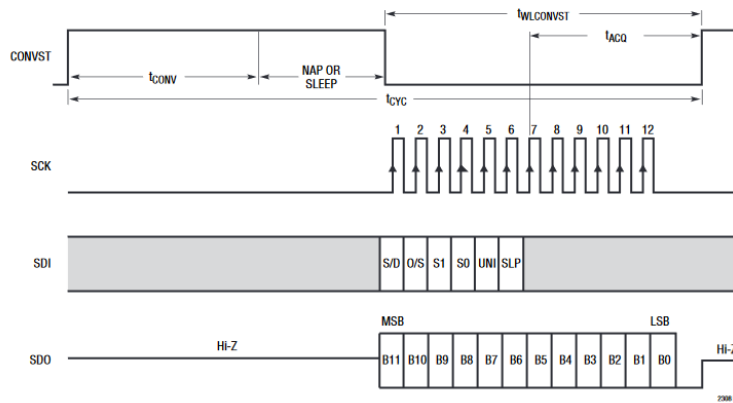


Figura 2.6 Temporización del LTC2308 con un pulso CONVST largo, [6].

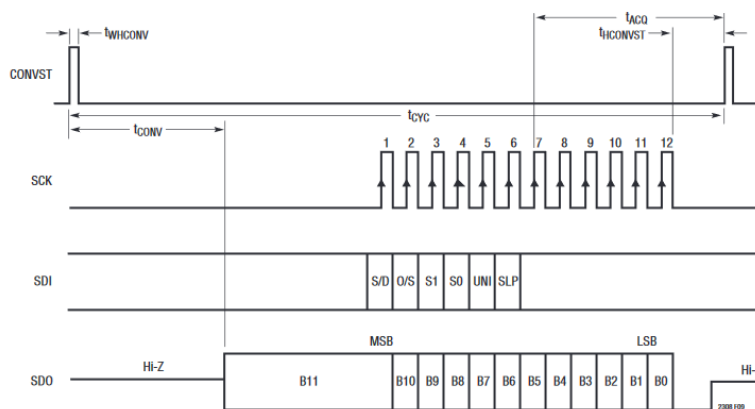


Figura 2.7 Temporización del LTC2308 con un pulso CONVST corto, [6].

2.3.3 DESARROLLO DE LA ARQUITECTURA PARA CONVERSIÓN

Para el desarrollo de la arquitectura hardware correspondiente a la conversión, se desarrolló el programa que se encargara de otorgar el diagrama de tiempos de la Figura 2.7.

El código del programa base del convertidor analógico digital tiene por entidad 4 entradas y 5 salidas las cuales se pueden observar en la Figura 2.8. Las 4 entradas son `clk`, `measure_start`, `measure_ch` y `ADC_DOUT`; `clk` es una entrada de un bit y su función es la de ser la base de tiempo del ADC, `measure_start` es una entrada de un bit cuya función es iniciar una conversión para el ADC, `measure_ch` es una entrada de 3 bits la cual indica el canal en el cuál el ADC realizará la lectura y `ADC_DOUT` la cual es una de las interfaces del ADC LTC2308 con el FPGA y es la encargada de portar el dato convertido de manera serial. En cuanto a las salidas se cuenta con 5, las cuales son: `measure_done`, `measure_dataread`, `ADC_CS_N`, `ADC_SCLK` y `ADC_DIN`; `measure_done` es una salida de un bit y sirve para indicar cuando se tiene un dato de salida valido, `measure_dataread` es una salida de 12 bits la cual porta el dato que se obtuvo de la conversión del ADC, `ADC_CS_N`, `ADC_SCLK` y `ADC_DIN` son interfaces que tiene el FPGA con el ADC LTC2308, `ADC_CS_N` indica cuando está habilitado el ADC, `ADC_SCLK` es la entrada de reloj serial para el ADC, `ADC_DIN` es una señal en la cual se debe colocar la palabra de control donde se indica el modo de operación y el canal de medición.

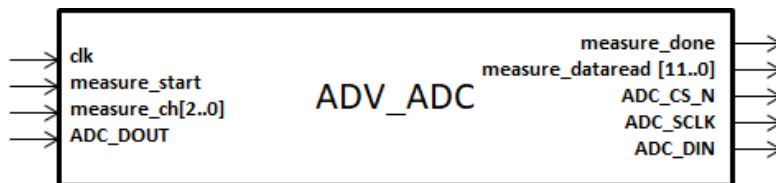


Figura 2.8 Entidad del código `ADV_ADC`.

El código comienza por declarar unas constantes las cuales sirven para delimitar el número de bits en la salida, el número de bits en la palabra de control, el número de canales utilizables, y los tiempos (en ciclos de reloj) en que se realizan las etapas del diagrama de tiempos en la Figura 2.7. Para los tiempos se selecciona primero el tiempo en que la salida `ADC_CS_N` se encuentra en alto ('1'), este tiempo debe ser al menos de 20ns, debido a que se utilizará un acumulador de fase para poder variar el reloj de entrada con el fin de variar también la frecuencia de muestreo, al usar un reloj de 50MHz la frecuencia máxima que se puede obtener con el acumulador de fase es la mitad (25MHz). Un ciclo de 25MHz equivale a 40ns, por lo que al ser esta la frecuencia

máxima que se tendrá por entrada de reloj, con un ciclo bastará para que ADC_CS_N esté en alto ya que pasará 40 ns en alto, en el diagrama de tiempos de la Figura 2.7 este tiempo corresponde a “ t_{WHCONV} ”. El siguiente tiempo que se selecciona es aquel que ocurre desde que se inicia la conversión hasta que se activa el reloj serial, en el diagrama de tiempos de la Figura 2.7 es llamado “ t_{CONV} ”, este tiempo es el que el ADC se tarda en convertir el dato de la medición anterior y en lo que está listo para mandarse al ADC, este tiempo normalmente tarda 1.3 μ s y máximo puede tardar 1.6 μ s (acorde a la hoja de datos [6]), en este caso se puso a 35 ciclos lo que equivale a un tiempo de 1.4 μ s. El siguiente tiempo a configurar es el tiempo muerto que hay después de que se obtienen los 12 bits de la conversión, en este caso se establece en 0, ya que no se desea alentar el proceso si no que se requiere hacerlo lo más rápido posible. Con esos tiempos definidos ya se puede proseguir a continuar con el código.

El código se divide en 8 procesos trabajando al mismo tiempo los cuales son:

- Reseteo de las variables para nueva medición.
- Contador.
- Estado de ADC_CS_N.
- Habilitación del reloj serial.
- Obtención de los 12 bits de lectura.
- Habilitar bandera de finalización de proceso.
- Selección del canal.
- Corrimiento de bits para la palabra de control.

El reseteo de variables funciona por medio de una señal llamada pre_measure_start, la cual en cada ciclo de reloj toma el valor de measure_start, de esta forma se detecta cuando measure_start pasa de bajo a alto ya que pre_measure_start actualiza su valor en cada ciclo de reloj, entonces existirá un momento en el que pre_measure_start esté en bajo y measure_start en alto.

El contador es un contador de ciclos de reloj, este inicia en 0, se incrementa en 1 y termina cuando se haya llegado al valor de la constante t_{DONE} , la cual representa el tiempo en ciclos de reloj que se tarda el proceso completo. Este contador se utiliza en

los otros procesos como medida de tiempo, para saber cuántos ciclos han pasado y que procesos deben realizarse en su tiempo adecuado.

ADC_CS_N se habilita según el contador y el tiempo que se haya predeterminado con anterioridad.

El reloj serial tiene una habilitación y este solo se presenta cuando el tiempo t_{CONV} haya transcurrido y se deshabilita cuando se hayan leído los 12 bits de lectura, por lo que dura habilitado 12 ciclos de reloj. El reloj serial es el mismo que el reloj maestro del sistema cuando se encuentra habilitado.

Para obtener los 12 bits de lectura se realiza un corrimiento en una señal, la cual va tomando los valores del ADC_DIN y los almacena en dicha señal, luego dicha señal es asignada a la salida donde se encuentran dichos 12 bits.

La salida de dato valido se pone en alto cuando el contador llega al tiempo t_{DONE} indicando que se realizó una conversión y el dato en la salida es un dato valido.

Cuando el reseteo de variables se realiza se crea la palabra de control que se le mandará de manera serial al ADC LTC2308, pero para obtener dicha palabra de control es necesario indicarle en que canal se realizará la medición, es por eso que se lee el canal que se indica en la entrada y se realiza la palabra de control que se mandará.

Finalmente, el último proceso que se realiza es el corrimiento de bits para la palabra de control que se mandará a través del ADC_DOUT, la cual se hace con habilitaciones que se activan y desactivan según el valor que lleve el contador.

Luego se realizó la simulación del código, con el fin de verificar que su funcionamiento sea el mismo que el de la Figura 2.7.

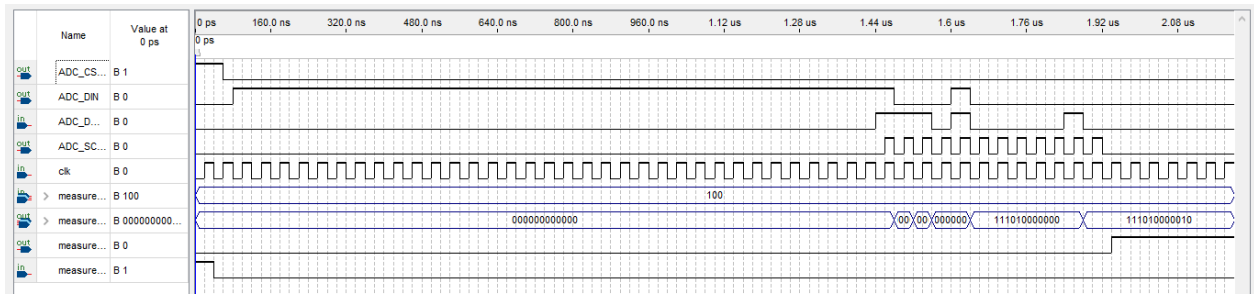


Figura 2.9 Simulación del código ADV_ADC.

La Figura 2.9 es muy similar a la Figura 2.7 como se podrá apreciar, de ella se pueden obtener también los tiempos en los que funciona el sistema y como se puede observar, pasan 48 ciclos de reloj para poder realizar una conversión, de los cuales 1 sirve para el reseteo de las variables, 35 son para el t_{CONV} y 12 en los que se guarda el valor leído en la salida `measure_dataread`.

El sistema para el que se utilizará este código requiere de 4 canales, es por eso que se creó un código el cual es una máquina de estados que coordina el inicio y la adquisición de un dato valido para el sistema de 4 canales.

La entidad del programa anterior presenta 3 entradas y 8 salidas, la cual se puede observar en la Figura 2.10. En cuanto a las entradas se tiene `CLOCK`, `RESET_N` y `ADC_DOUT`. `CLOCK` es la entrada de reloj, el `RESET_N` es una señal de reseteo la cual reiniciará la máquina de estados encargada de iniciar las conversiones para los canales seleccionados. Y `ADC_DOUT` es una de las interfaces del ADC con el FPGA. Las salidas son: `output`, `output2`, `output3`, `output4`, `PUSH`, `ADC_SCLK`, `ADC_CS_N` y `ADC_DIN`. `Output`, `output2`, `output3` y `output4` son salidas de 12 bits las cuales portan las lecturas realizadas por el ADC del canal 6, 4, 2 y 0 respectivamente, estos canales fueron seleccionados de esta manera ya que físicamente están consecutivos. `PUSH` es una señal de reloj generada a partir de la máquina de estados, su función es indicar cuando se realizó una conversión de los 4 canales. `ADC_SCLK`, `ADC_CS_N` y `ADC_DIN` son interfaces del ADC con el FPGA.

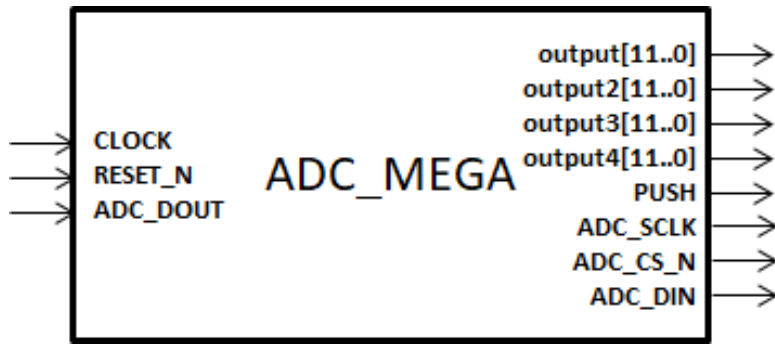


Figura 2.10 Entidad del programa ADC_MEGA.

En este código se instancia el código llamado ADV_ADC, y lo que se hace es crear una máquina de estados que coordine su inicio y obtención de un dato valido. El diagrama de estados de la máquina de estados utilizada en el código ADC_MEGA se ilustra en la Figura 2.11, en la Tabla 2.1 se resume la FSM por estados.

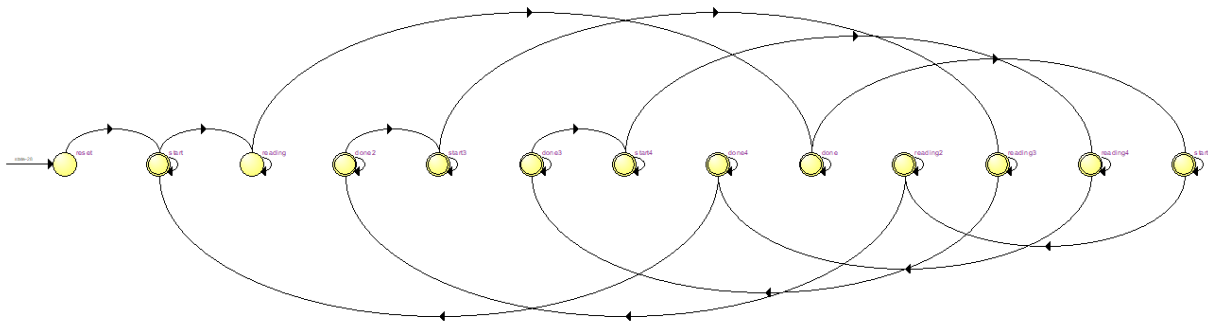


Figura 2.11 Diagrama de estados de la FSM utilizada en el código ADC_MEGA.

Tabla 2.1 Resumen de la FSM utilizada en el código ADC_MEGA.

Estado	Estado próximo	Cambio de estado	Utilidad del estado
reset	start	Flanco de bajada del reloj	Es el estado de reset, cuando RESET_N vale 0 se va a este estado sin importar en cual estaba antes.

start	reading	Flanco de bajada de reloj	Sirve para iniciar la conversión en el canal 0.
reading	done	Flanco de bajada del reloj y salida valida del ADC en '1'	Se encuentra en este estado durante el tiempo de conversión del canal 0.
done	start2	Flanco de bajada del reloj y salida valida del ADC en '1'	Almacena en una localidad de una memoria RAM el valor de la conversión hecha para el canal 6.
Start2	reading2	Flanco de bajada del reloj	Se inicia la conversión para el canal 2 y se manda a nivel alto la señal que indica que los 4 canales han realizado su conversión.
Reading2	done2	Flanco de bajada del reloj y salida valida del ADC en '1'	Se encuentra en este estado durante el tiempo de conversión del canal 2.
Done2	start3	Flanco de bajada del reloj y salida valida del ADC en '1'	Almacena en una localidad de una memoria RAM el valor de la conversión hecha para el canal 0.
Start3	reading3	Flanco de bajada de reloj	Sirve para iniciar la conversión en el canal 4.
Reading3	done3	Flanco de bajada del reloj y salida valida del ADC en '1'	Se encuentra en este estado durante el tiempo de conversión del canal 4.

Done3	start4	Flanco de bajada del reloj y salida valida del ADC en '1'	Almacena en una localidad de una memoria RAM el valor de la conversión hecha para el canal 2.
Start4	reading4	Flanco de bajada del reloj	Sirve para iniciar la conversión en el canal 6.
Reading4	done4	Flanco de bajada del reloj y salida valida del ADC en '1'	Se encuentra en este estado durante el tiempo de conversión del canal 6
done4	start	Flanco de bajada del reloj y salida valida del ADC en '1'	Almacena en una localidad de una memoria RAM el valor de la conversión hecha para el canal 4.

Cuando se terminan las 4 conversiones se habilita una señal que manda a cada salida un nuevo valor de conversión.

Los cambios de estados retrasan la conversión 2.5 ciclos, los cuales al sumarse a los 48 ciclos que tarda la conversión dan 50.5 ciclos de reloj por canal, pero como se multiplexan 4 canales, se obtiene la conversión de los cuatro cada 202 ciclos de reloj.

Para poder controlar la frecuencia de muestreo es necesario cambiar la base de tiempo que se utiliza, es decir la frecuencia del reloj, para realizar esto existe un algoritmo llamado acumulador de fase.

El acumulador de fase, es un sistema digital que a cada pulso activo de reloj suma el valor de la fase P, con el valor acumulador previamente A y lo vuelve a almacenar en A para producir la salida Q. De esta forma, la señal Q tiene un comportamiento de rampa digital y cuya frecuencia está dada por:

$$f_Q = \frac{P}{2^n} f_{CLK} \quad (1)$$

La característica principal de la ecuación 1 es que se puede producir un generador de base de tiempo, cuya frecuencia de salida sigue un patrón directamente proporcional a la frecuencia base de reloj. Esta característica puede ser comparada con la generación de base de tiempo por medio de un contador, donde la frecuencia es inversamente proporcional al módulo N del mismo:

$$f_{CNT} = \frac{1}{N} f_{CLK} \quad (2)$$

Las frecuencias que se pueden obtener con un contador módulo N, no se encuentran espaciadas de manera uniforme, mientras que las frecuencias obtenidas con el acumulador de fase si se encuentran espaciadas de manera uniforme, [7].

El acumulador de fase tiene un problema, el cual es que su frecuencia máxima obtenible es la mitad de la del reloj de entrada. Al usar el reloj de 50MHz que incluye la tarjeta DE1 SoC, la frecuencia del reloj de entrada se podrá variar de 0 a 25 MHz.

La entidad del programa que une al acumulador de fase con el controlador de ADC indica 4 entradas y 8 salidas. Las entradas son clk, Fs, reset_n y ADC_DOUT y las salidas son ADC_CS_N, ADC_DIN, ADC_SCLK, PUSH, output, output2, output3 y output4. Las entradas y salidas son las mismas que en el código ADC_MEGA, con la excepción de que la entrada de reloj (clk) es para el acumulador de fase y Fs es la entrada de control del acumulador de fase.

El funcionamiento del código es sencillo, el acumulador de fase se utiliza para generar una salida de reloj cuya frecuencia depende del valor de entrada de control del acumulador de fase. Dicha salida de reloj generada con el acumulador de fase se conecta como entrada de reloj a al módulo de control del ADC tal como se muestra en la Figura 2.12.

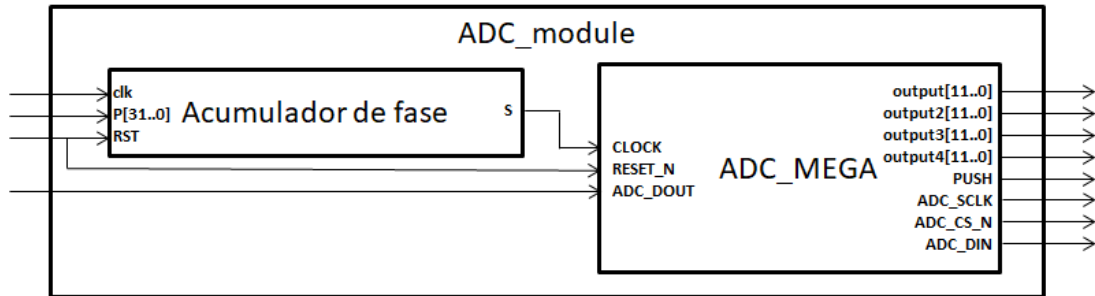


Figura 2.12 Diagrama de bloques del código ADC_module.

Como se mencionó anteriormente, utilizando el acumulador de fase para cambiar la frecuencia de la entrada de reloj del módulo del ADC, se puede variar de 0 a 25 MHz, lo que implicaría que la entrada de reloj máxima es de 25MHz, si el realizar una conversión para los 4 canales implica 202 ciclos de reloj, eso quiere decir que la frecuencia de muestreo máxima está dada por:

$$f_{s_{\max}} = \frac{25\text{MHz}}{202} = 123.762 \text{ kHz} \quad (3)$$

Ahora solo se debe caracterizar la entrada P del acumulador de fase (encargada de variar la frecuencia de muestreo) con respecto a la frecuencia de muestreo deseada.

Si se despeja la ecuación 1 en términos de P se obtiene:

$$P = \frac{2^n}{f_{\text{CLK}}} f_Q \quad (4)$$

La frecuencia de muestreo en términos de la frecuencia de entrada está dada por:

$$f_s = \frac{f_Q}{202} \quad (5)$$

Si se despeja f_Q de la ecuación 5 se obtiene:

$$f_Q = 202f_s \quad (6)$$

Ahora, si se reemplaza la ecuación 6 en la ecuación 4, se obtiene una ecuación que relaciona la entrada del acumulador de fase con la frecuencia de muestreo deseada:

$$P = \frac{(2^n)(202)}{f_{CLK}} f_s \quad (7)$$

donde para la ecuación 7: n es el número de bits de la entrada P (32 bits) y f_{CLK} es la entrada de reloj principal (50 MHz). Reemplazando estos valores en la ecuación 7 se obtienen las siguientes ecuaciones:

$$P = \frac{(2^{32})(202)}{50 \times 10^6} f_s \quad (8)$$

$$P = 17351.66788 f_s \quad (9)$$

La ecuación 9 relaciona directamente la frecuencia de muestreo deseada con la entrada de control al acumulador de fase P, logrando así controlar la frecuencia de muestreo del ADC.

2.4 IMPLEMENTACIÓN DE LA TRANSFORMADA RÁPIDA DE FOURIER

2.4.1 TRANSFORMADA RÁPIDA DE FOURIER

La serie de Fourier, es la representación matemática básica para señales periódicas, la cual es una sumatoria de componentes armónicamente relacionadas a señal original. La expresión que relaciona las armónicas c_k (componentes exponenciales) con la señal $x(t)$ es:

$$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{j2\pi k F_0 t} \quad (10)$$

donde: $T_0 = \frac{1}{F_0}$, es el periodo de la componente fundamental de la señal. Esta expresión se conoce como la ecuación de síntesis. La expresión correspondiente a la obtención de los coeficientes, se conoce como ecuación de análisis y es:

$$c_k = \frac{1}{T_p} \int_{T_p} x(t) e^{-j2\pi k F_0 t} dt \quad (11)$$

En general, los coeficientes de Fourier c_k son valores complejos. Si la señal periódica es real, c_k y c_{-k} son complejos conjugados.

$$c_k = |c_k| e^{j\theta_k} \quad (12)$$

$$c_{-k} = |c_k| e^{-j\theta_k} \quad (13)$$

Como resultado, la ecuación 10 puede ser representada de la forma:

$$x(t) = c_0 + 2 \sum_{k=1}^{\infty} |c_k| \cos(2\pi k F_0 t + \theta_k) \quad (14)$$

Utilizando identidades trigonométricas, la ecuación 14 puede reescribirse como:

$$x(t) = a_0 + \sum_{k=1}^{\infty} (a_k \cos 2\pi k F_0 t - b_k \sen 2\pi k F_0 t) \quad (15)$$

donde:

$$a_0 = c_0 \quad (16)$$

$$a_k = 2|c_k| \cos \theta_k \quad (17)$$

$$b_k = 2|c_k| \sen \theta_k \quad (18)$$

Considerando ahora que la señal $x(t)$ es aperiódica, la representación de la señal en componentes frecuenciales está dada por:

$$x(t) = \int_{-\infty}^{\infty} X(F) e^{j2\pi F t} dt \quad (19)$$

Esta expresión se conoce como la ecuación de síntesis. Al igual que la serie de Fourier, tiene se correspondiente ecuación de análisis, o también llamada Transformada de Fourier de $x(t)$, dada por:

$$X(F) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi Ft} dt \quad (20)$$

Debido a que un procesador digital no puede manipular directamente datos analógicos, es necesario discretizar las señales, es decir, suponiendo una señal en tiempo continuo $x(t)$, se digitalizará mediante un muestreador con un periodo T o una frecuencia de muestreo $\omega_s = 2\pi/T$, para obtener la señal en tiempo discreto $x[n] = x(nT)$.

Por tanto, las expresiones correspondientes a la serie de Fourier en tiempo discreto son: ecuación de síntesis (21) y ecuación de análisis (22):

$$x(n) = \sum_{k=0}^{N-1} c_k e^{j2\pi kn/N} \quad (21)$$

$$c_k = \frac{1}{N} \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N} \quad (22)$$

donde: N es el periodo de $x(n)$; se generan N armónicas relacionadas a funciones exponenciales.

De la misma forma que en tiempo continuo, las ecuaciones correspondientes a la transformada de Fourier para señales aperiódicas en tiempo discreto son:

$$X(\omega) = \sum_{k=-\infty}^{\infty} x(n) e^{-j\omega n} \quad (23)$$

Y:

$$x(n) = \frac{1}{2\pi} \int_{2\pi} X(\omega) e^{j\omega n} d\omega \quad (24)$$

Llamadas ecuaciones de análisis y de síntesis, correspondientemente, y referidas como Transformada Discreta de Fourier (*DFT, Discrete Fourier Transform*).

Cuando la secuencia $x(n)$ tiene una duración finita $L \leq N$, la señal resultante, es una porción de la señal original y consecuentemente, las muestras en frecuencia $X(2\pi k/N)$,

para $k = 0, 1, \dots, N - 1$, únicamente representan la duración finita de la secuencia $x(n)$. La DFT para una secuencia $x(n)$ de longitud L esta dada por:

$$X(\omega) = \sum_{n=0}^{L-1} x(n)e^{-j\omega n} \quad 0 \leq \omega \leq 2\pi \quad (25)$$

Uno de los problemas computacionales de la DFT es calcular la secuencia $\{X(k)\}$ de N números de valor complejo, dada una secuencia de datos $x(n)$ de longitud N , acorde a la fórmula:

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{kn} \quad 0 \leq \omega \leq 2\pi \quad (26)$$

donde,

$$W_N^n = e^{-j2\pi n/N} \quad (27)$$

es conocido como el factor de rotación (Twiddle Factor). Por lo que la DFT puede obtenerse utilizando las propiedades de simetría y periodicidad del factor W_N^n , para el cálculo de la misma. Dado que el cálculo directo de la DFT de N puntos requiere $N(N - 1)$ adiciones complejas N^2 multiplicaciones complejas. La complejidad puede ser reducida al orden de $N \log_2 N$ con el algoritmo conocido como Transformada Rápida de Fourier (*FFT, Fast Fourier Transform*).

Para la simplificación, entre algunos de los algoritmos utilizados en la FFT, se encuentra el conocido como “*algoritmo de decimación en tiempo*” el cual asume que N es potencia de 2. Para evidenciar la simplificación, a continuación se desglosa la teoría básica del mismo. Inicialmente se realiza la separación de la sumatoria en sumas pares e impares de n está dada por:

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n)W_N^{2nk} + \sum_{n=0}^{\frac{N}{2}-1} x(2n + 1)W_N^{(2n+1)k} \quad (28)$$

Para $k = 0, 1, \dots, N - 1$.

Haciendo que los puntos pares sean denotados como:

$$a[n] = x[2n] \quad (29)$$

Para $n = 0, 1, \dots, \frac{N}{2} - 1$

Y los impares como:

$$b[n] = x[2n + 1] \quad (30)$$

Para $n = 0, 1, \dots, \frac{N}{2} - 1$

Además, se puede observar que $W_N^2 = W_{\frac{N}{2}}^2$, de esta forma, la ecuación de 27 puede ser reescrita como:

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} a[n]W_{N/2}^{nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} b[n]W_{N/2}^{nk} \quad (31)$$

Para $k = 0, 1, \dots, N - 1$.

Asignando A_k y B_k como la DFT de $N / 2$ puntos de $a[n]$ y $b[n]$ tal que estas DFT's tengan un periodo de $N / 2$. Con estas definiciones la ecuación 31 se transforma a:

$$X(k) = A_k + W_N^k B_k \quad (32)$$

Para $k = 0, 1, \dots, N - 1$.

Observado el factor de rotación (ecuación 27) se obtiene que: $W_N^{N/2} = -1$. Por lo tanto, la ecuación de $X(k)$ puede ser separada de la siguiente forma:

$$X(k) = A_k + W_N^k B_k \quad (33)$$

Para $k = 0, 1, \dots, \frac{N}{2} - 1$

$$X\left(k + \frac{N}{2}\right) = A_k - W_N^k B_k \quad (34)$$

Para $k = 0, 1, \dots, \frac{N}{2} - 1$

Con las ecuaciones 33 y 34, se muestra que es posible calcular una DFT de N puntos, combinando un par de DFT's de $N / 2$ puntos, este cálculo es llamado "Mariposa FFT" (FFT Butterfly) debido a la forma gráfica del flujo de operaciones. De esta manera la DFT de N puntos puede ser obtenida con $N/2$ multiplicaciones complejas para calcular los factores $W_N^k B_k$, N adiciones complejas desde un par de DFT's de $N/2$ puntos [8].

2.4.2 MEMORIAS FIFO

Una memoria FIFO está formada por una disposición de registros de desplazamiento. El término FIFO hace referencia al funcionamiento básico de este tipo de memoria, en la que el primer bit de datos que se escribe es el primero que se lee.

En la Figura 2.13 se ilustra una diferencia importante entre un registro de desplazamiento convencional y un registro FIFO. En un registro convencional, un bit de datos se desplaza a través del registro sólo cuando se introducen nuevos datos; en un registro FIFO, un bit de datos atraviesa el registro hasta situarse en la posición de bit más a la derecha que esté vacía.

La Figura 2.14 es el diagrama de bloques de una memoria serie FIFO. Esta memoria en particular tiene cuatro registros de datos serie de 64 bits y un registro de control de 64 bits (registro de marca). Cuando los datos se introducen mediante un impulso de desplazamiento de entrada, automáticamente, bajo el control del registro de marca, se mueven a la posición vacía más próxima a la salida. Los datos no pueden avanzar a las posiciones que están ocupadas. Sin embargo, cuando un bit de datos se desplaza mediante un impulso de desplazamiento de salida, los bits de datos que están en los registros automáticamente se mueven a la posición siguiente hacia la salida. En una memoria FIFO asíncrona, los datos se desplazan hacia fuera independiente-mente de la entrada de datos, utilizando dos relojes separados.

Registro de desplazamiento convencional					
Entrada	X	X	X	X	Salida
0	0	X	X	X	→
1	1	0	X	X	→
1	1	1	0	X	→
0	0	1	1	0	→

X = bits de datos desconocidos.

En un registro de desplazamiento convencional, los datos permanecen a la izquierda hasta que son desplazados por medio de datos adicionales.

Registro de desplazamiento FIFO					
Entrada	—	—	—	—	Salida
0	—	—	—	0	→
1	—	—	1	0	→
1	—	1	1	0	→
0	0	1	1	0	→

— = posiciones vacías.

En un registro de desplazamiento FIFO, los datos "van cayendo" hacia la d

Figura 2.13 Comparación del funcionamiento de un registro convencional y uno FIFO, [9].

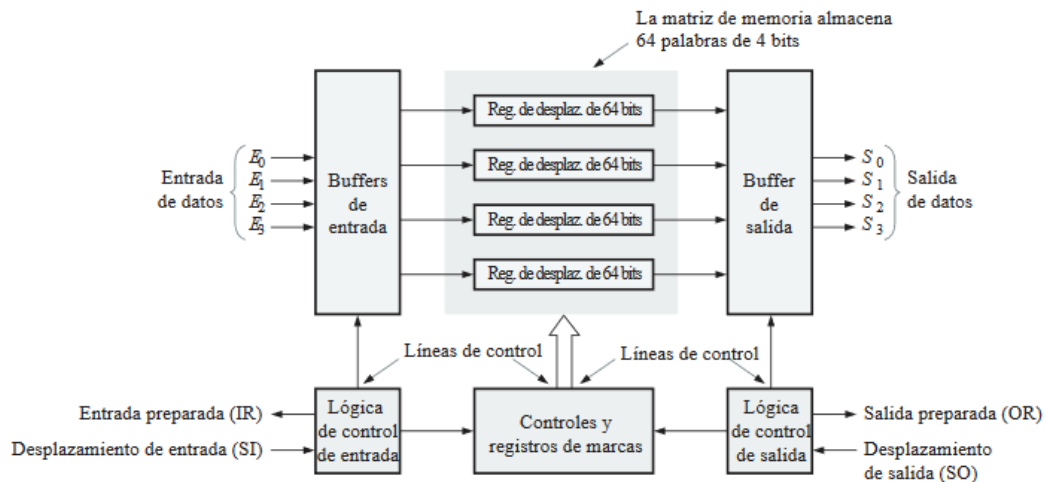


Figura 2.14 Diagrama de bloques de una memoria FIFO serie típica, [9].

Un área de aplicación importante del registro FIFO es el caso en que dos sistemas con velocidades diferentes tienen que comunicarse. Los datos pueden entrar en un registro

FIFO a una velocidad y salir a otra velocidad distinta. La Figura 2.15 muestra cómo debe emplearse un registro FIFO en estas situaciones.

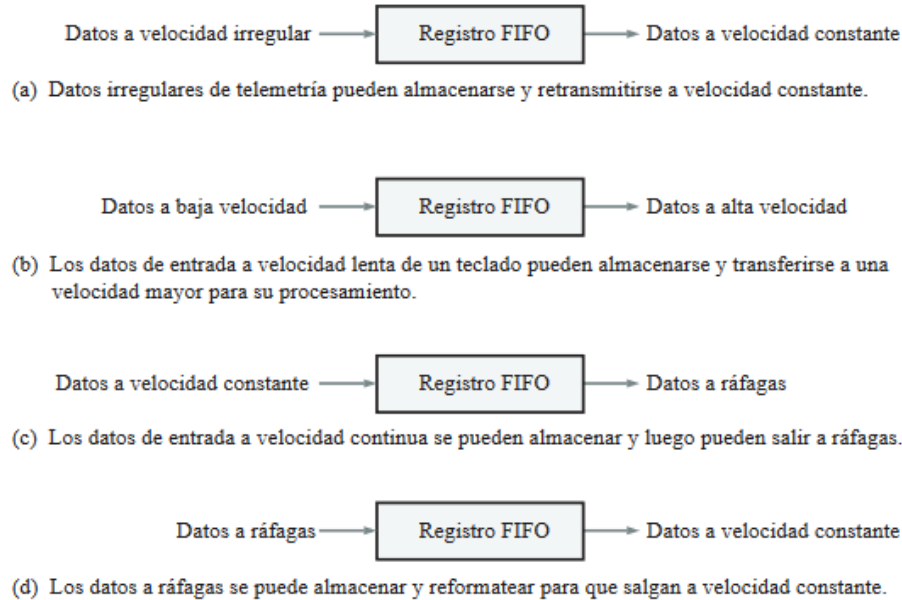


Figura 2.15 Ejemplos de registro FIFO para aplicaciones de control de la velocidad de transmisión de datos, [9].

2.4.3 DESARROLLO DE LA ARQUITECTURAS PARA OBTENCIÓN DE LA FFT DE LA SEÑAL

La FFT fue obtenida con ayuda del HDL Coder que tiene Simulink de Matlab. Desde Simulink se realizó un modelo donde se utilizaron los bloques: FFT HDL Optimized y Complex to Magnitude-Angle HDL Optimized. El bloque FFT HDL Optimized ayuda para poder realizar la transformada rápida de Fourier, pero su salida la da como un número complejo, lo que interesa conocer para el sistema a realizar es la magnitud de dicho número complejo el cual se obtiene con el bloque Complex to Magnitude-Angle HDL Optimized el cual por medio del algoritmo CORDIC obtiene la magnitud y el ángulo de un número complejo, [10]. El diseño del bloque en Simulink se puede apreciar en la Figura 2.16.

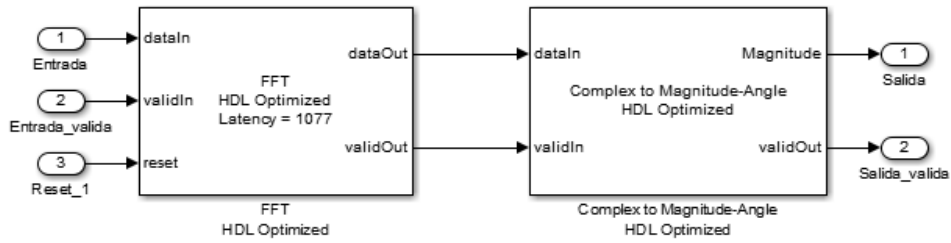


Figura 2.16 Modelo de Simulink.

Por medio de la herramienta HDL Coder que permite generar el código VHDL de un modelo de Simulink, se obtuvo el código de la FFT en VHDL. Una vez con los archivos generados con el HDL Coder se puede abrir un nuevo proyecto de Quartus y añadir los archivos anteriores al proyecto. El código de más alta entidad es aquel que tenga el mismo nombre con el que se guardó el archivo .slx (El modelo de Simulink), en este caso se guardó con el nombre “fourier_mag_512.slx” ya que se generó una FFT con un tamaño de 2^9 (512) entonces el archivo donde se encuentra la entidad de más alto nivel se llamará “fourier_mag_512.vhd”, cuyas entradas y salidas se muestran en la Figura 2.17.

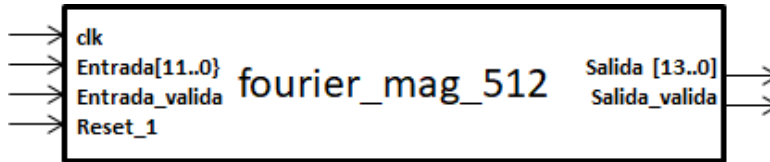


Figura 2.17 Entidad del programa generado con HDL Coder.

El código anterior cuenta con cuatro entradas (clk, Entrada, Entrada_valida y Reset_1) y dos salidas (Salida y Salida_Valida). La entrada clk es la entrada de reloj con la que operará la transformada de Fourier, ya que requiere un tiempo para calcular la transformada de Fourier de los 512 datos de entrada. Entrada es una entrada de 12 bits la cual son los 512 datos a los cuales se les realizará el algoritmo de la FFT, son 12 bits ya que la entrada provendrá de un ADC cuya salida de conversión la realiza a 12 bits. Entrada_valida funciona como entrada de habilitación, si esta entrada se encuentra en alto y hay un flanco ascendente en la entrada de reloj entonces la entrada se considera

valida. Reset_1 es la entrada de reseteo para el programa. Salida es una salida de 14 bits en ellos va el resultado de la magnitud del resultado de la transformada rápida de Fourier. Salida_Valida es una salida que indica cuando Salida ya tiene resultados válidos.

Para ver el funcionamiento del código generado se realizó una simulación en la cual se ingresó por valores de entrada un contador con el fin de conocer en cuantos ciclos de reloj se tarda en sacar el resultado, dicha simulación se puede observar en la Figura 2.18. Los resultados que dio la FFT generada se compararon con los dados por un script hecho en Matlab.

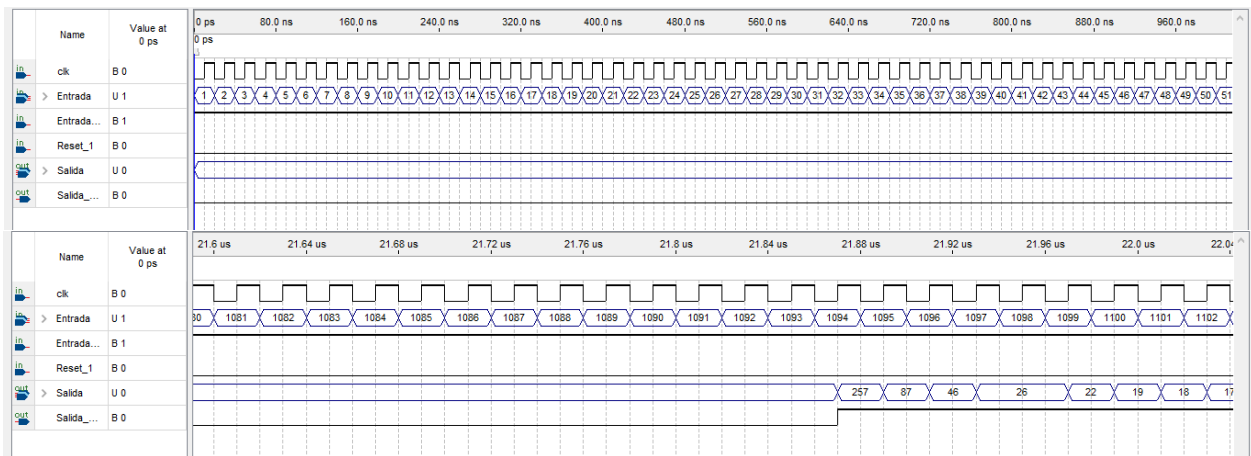


Figura 2.18 Simulación del código generado con HDL Coder,

Como se indica en la Figura 2.18, la FFT tarda 1094 ciclos de reloj para obtener un resultado válido. Lo que prosiguió fue verificar que los resultados obtenidos fueran correctos. Al ser la entrada un contador ascendente los 512 datos a los cuales se les calculo su FFT fue a una rampa que inicia en el 1 y termina en el 512. Si el reloj de entrada en la simulación es uno de 50MHz, entonces la ecuación de la señal de entrada en tiempo continuo sería:

$$S(t) = 50 \times 10^6 t \quad (35)$$

Se creó un script de Matlab para obtener la FFT de la ecuación 35 si se discreta con una frecuencia de muestreo de 50MHz.

En el script se realizan algunas instrucciones para poder observar de manera gráfica, tanto la ecuación como la FFT y se puede observar en la Figura 2.19. En la Figura 2.20 se observa la comparación entre los resultados que otorga el código en VHDL con los que da el script de Matlab y se observa que no hay mucha diferencia, en base a los valores obtenidos se encuentra con un error aproximado del 12.69%.

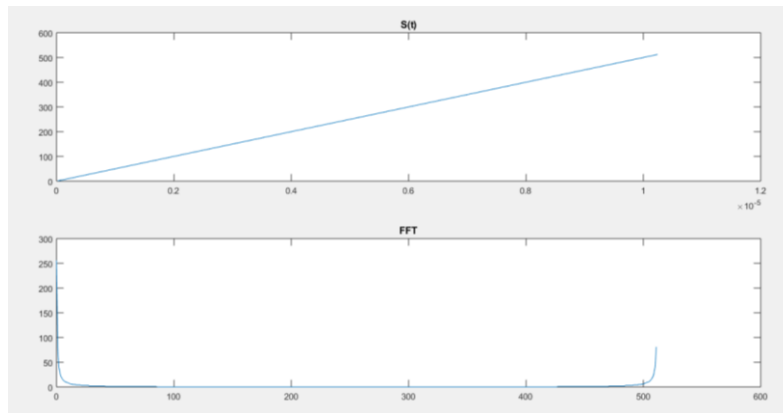


Figura 2.19 Señal de entrada y su espectro de frecuencias.

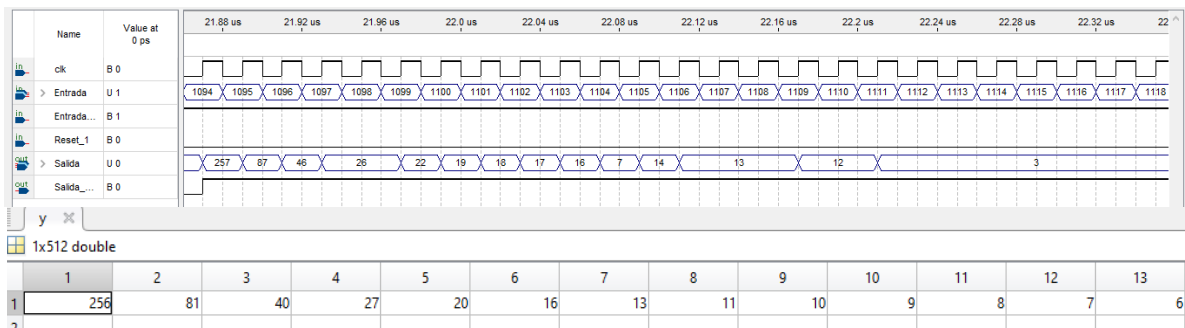


Figura 2.20 Comparación de la salida del código en VHDL contra los resultados obtenidos utilizando la herramienta Matlab.

Se diseñó un programa en VHDL con el fin de unir el controlador del ADC LTC2308 (que es el que trae instalado la tarjeta DE1 SoC) con la FFT para obtener el espectro de frecuencias de una señal adquirida con dicho ADC.

La entidad del sistema tiene 6 entradas (clk, reset_n, Fs, FFT_en, POP_FFT y ADC_DOUT) y 13 salidas (output, output2, output3, output4, output_fft, output2_fft, output3_fft, output4_fft, PUSH, FFT_done, ADC_CS_N, ADC_DIN y ADC_SCLK) y se

puede observar en la Figura 2.21. Algunas de estas entradas y salidas pertenecen al controlador del ADC tal y como lo son `reset_n`, `Fs`, `output`, `output2`, `output3`, `output4`, `PUSH`, `ADC_DOUT`, `ADC_CS_N`, `ADC_DIN` y `ADC_SCLK`. La entrada `reset_n` funciona para resetear el controlador del ADC; `Fs` es la entrada que funciona para controlar la frecuencia de muestreo del ADC; `output`, `output2`, `output3` y `output4` son las salidas de 12 bits donde se encuentran las lecturas realizadas en los canales 6, 4, 2 y 0 del ADC respectivamente; `PUSH` es una señal de reloj que lleva la misma frecuencia que la frecuencia de muestreo; `ADC_DOUT`, `ADC_CS_N`, `ADC_DIN` y `ADC_SCLK` son interfaces del ADC LTC 2308 con el FPGA.

La entrada `clk` es la entrada de reloj maestra, esta entrada funciona tanto para la FFT como para el ADC; `FFT_en` es una entrada que sirve para habilitar la FFT cuando se solicite obtenerla; `POP_FFT` es una señal la cual permite obtener un valor de la FFT almacenado una memoria FIFO. Las salidas: `output_fft`, `output2_fft`, `output3_fft` y `output4_fft` son las salidas de la FIFO y corresponden a un valor de la FFT de los canales 6, 4, 2 y 0 del ADC respectivamente; `FFT_done` es una señal que indica cuando se terminó de calcular la FFT y se pueden extraer de su memoria FIFO los datos almacenados.

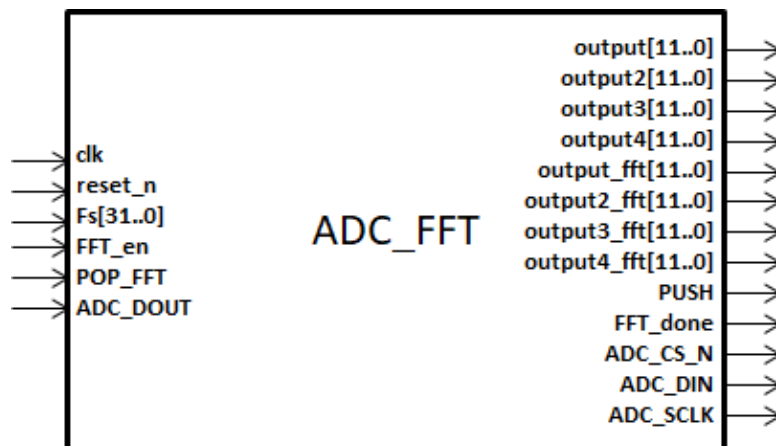


Figura 2.21 Entidad del programa ADC_FFT.

El programa funciona principalmente con 3 códigos: El controlador de ADC (`ADC_module` que se puede observar en la Figura 2.22), el código generado que

obtiene la FFT (fourier_mag_512 que se puede observar en la Figura 2.17), y una memoria FIFO asíncrona (aFifo que se puede observar en la Figura 2.23).

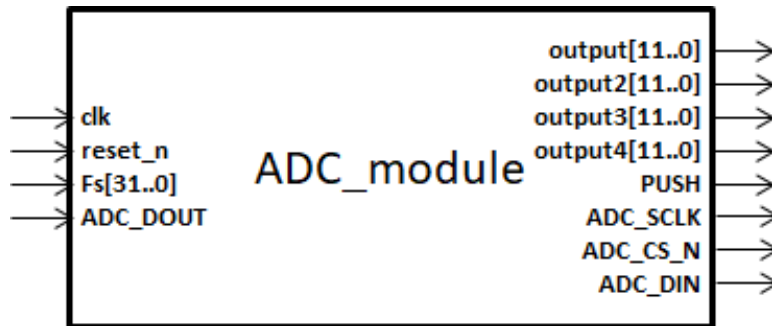


Figura 2.22 ADC_module.

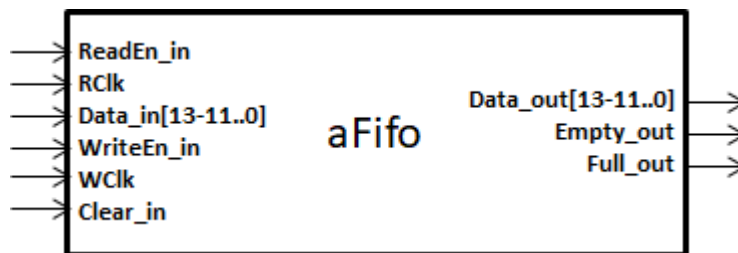


Figura 2.23 Memoria FIFO asíncrona.

El código funciona mandando las salidas ADC al módulo de la FFT, para que se calcule su espectro de cada salida, una vez terminada los resultados son enviados a una memoria FIFO asíncrona, una vez la memoria obtenga todos los resultados los resultados se podrán sacar de la FIFO con la entrada POP_FFT.

Las entradas de control de la FFT y de la FIFO son controladas según los ciclos de reloj que hayan pasado para esto se utiliza un contador. La primer entrada de control que se necesita ajustar es el reloj con el que trabaje la FFT, el programa está hecho para que una vez habilitada la FFT los primeros 512 ciclos trabaje con la salida PUSH del ADC la cual ingresara 512 datos que se obtengan del ADC los cuales serán los datos a los que se les saque la FFT, asegurándose que sean los datos a la frecuencia de muestreo conocida, después de los primeros 512 datos el reloj será el de 50MHz con el fin de que los cálculos y el llenado de la FIFO sean de una manera rápida.

Se tiene una señal que indica cuando la FFT cumplió con sus tres fases: obtención de datos, cálculo de la FFT y llenado de FIFO. Esto pasa después de 1606 ciclos 512 para obtener datos, 566 para el cálculo de la FFT, 16 para el cálculo de la magnitud de la FFT y 512 para almacenar los resultados de la FFT en la FIFO. Esta señal además de funcionar como bandera de que la FFT terminó habilita también la lectura en la FIFO para poder sacar los datos de ella.

Así como la FIFO no permite leer hasta que la FFT se termine también se cuenta con una entrada de habilitación para la escritura de la FIFO, y esta no se activa sino hasta 1084 ciclos de reloj, 10 antes ciclos antes de que salgan los resultados de la FFT, esto se hace así debido a que se detectó un error con los primeros datos que se ingresan a la FIFO según su simulación observada en la Figura 2.24.

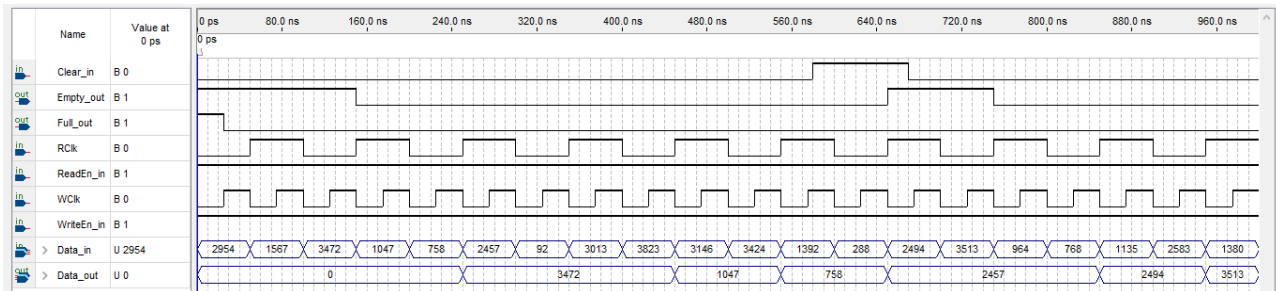


Figura 2.24 Simulación de la memoria FIFO.

Así como los resultados de la FFT se almacenan en una FIFO para con una señal de control (POP_FFT) poder sacar los datos, también es necesaria una para poder visualizar sin que se pierda ningún dato la señal que se obtiene del ADC.

En el código para colocarle una FIFO a cada salida de los canales del ADC se añaden solamente dos señales de entrada las cuales son las señales de control de la FIFO donde se almacenará la salida del ADC, estas son dos entradas y se llaman enable y POP, lo anterior puede apreciarse en la entidad del código mostrado en la Figura 2.25. POP es el reloj de lectura de las FIFO, cada que haya un flanco ascendente se sacara un valor de la FIFO; enable es la entrada que habilita la escritura y lectura de la FIFO.

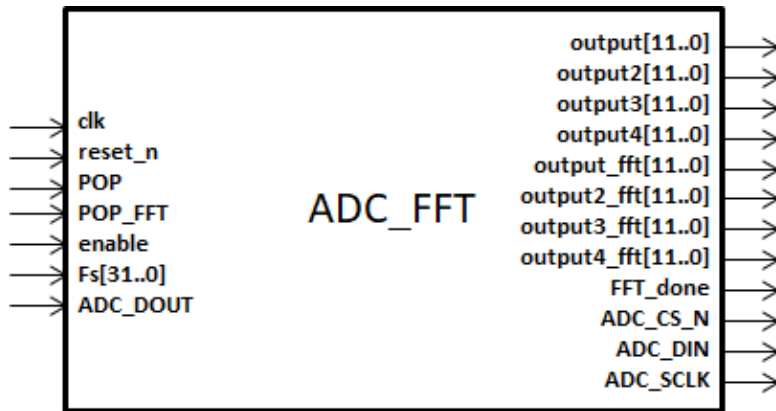


Figura 2.25 Entidad de ADC_FIFO.

Básicamente el programa une las salidas de las conversiones que hace el ADC con una FIFO, con el fin de poder ver los datos sin que necesariamente se vean a la frecuencia de muestreo. La frecuencia de muestreo está inmersa en la salida llamada PUSH en forma de reloj, esta salida se conecta como el reloj de escritura de la FIFO, la FIFO cuenta con entradas de habilitación por lo que mientras no esté activa no se podrá ni meter datos a la FIFO ni sacarlos.

Algo muy importante cuando se manejan memorias FIFO es el tamaño de la FIFO, dicho tamaño depende de las frecuencias de los relojes de escritura y de lectura. Más adelante se calculará el tamaño de la memoria, ya que se requiere la especificación de la frecuencia de lectura, aun cuando ya se conozca la de escritura (frecuencia de muestreo).

El sistema de adquisición de señales con la FFT implementada se puede observar en la Figura 2.26.

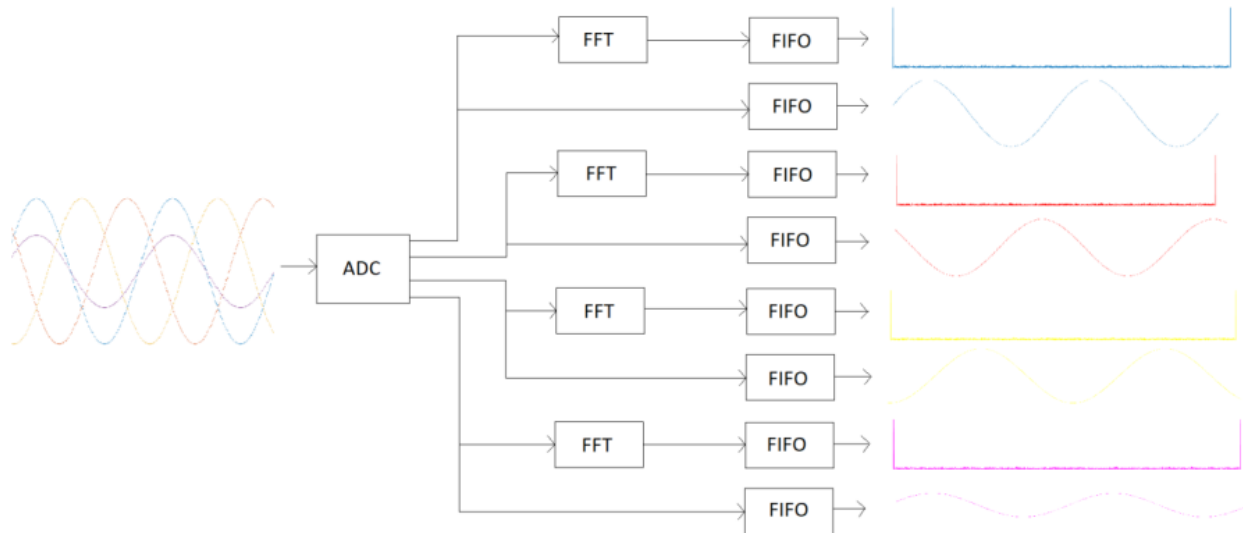


Figura 2.26 Arquitectura del sistema.

2.5 CIRCUITO DE ACONDICIONAMIENTO DE SEÑALES DE ENTRADA PARA EL ADC

2.5.1 ACONDICIONAMIENTO DE SEÑAL

En general, la señal de salida de un sistema de medición se debe procesar de una forma adecuada en sus distintas etapas: multiplexado, amplificación, filtrado y aislamiento. Debido a que la señal física medida puede ser muy pequeña, por lo cual resulta necesario amplificarla; podría contener interferencias que eliminar; no ser lineal y por ende, requerir una linealización, ser analógica y necesitar una digitalización; o viceversa; ser un cambio de voltaje y convertirla a un cambio en corriente de magnitud adecuada, entre otros casos. A estas modificaciones se refiere el concepto de acondicionamiento de señal, [11].

Los siguientes son algunos de los procesos que se pueden presentar en el acondicionamiento de una señal.

- a) Protección para evitar el daño al siguiente elemento.

- b) Convertir una señal en tipo de señal adecuada. Sería el caso cuando es necesario convertir una señal a un voltaje de corriente directa, a una corriente o presión.
- c) Obtener un nivel adecuado de la señal. En muchos casos es necesario amplificar una señal para que esta pueda ser leída.
- d) Eliminación o reducción de ruido. La forma más común es utilizando filtros.
- e) Manipulación de la señal. Por ejemplo, convertir una variable en una función lineal.

2.5.2 AMPLIFICADORES OPERACIONALES

El fundamento de numerosos módulos para el acondicionamiento de señales es el amplificador operacional. Este dispositivo amplificador de la diferencia de sus dos entradas, con una alta ganancia, es decir, una impedancia de entrada muy alta, (mayor a $1\text{ M}\Omega$) y una baja impedancia de salida (de 8 a $20\ \Omega$). Con estas características se deduce que las corrientes de entrada son prácticamente nulas y que tiene la característica de poder entregar corriente relativamente alta. Está disponible como circuito en chips de silicio. Tiene dos entradas: una inversora (-) y otra no inversora (+). La salida depende de cómo se hagan las conexiones de estas entradas.

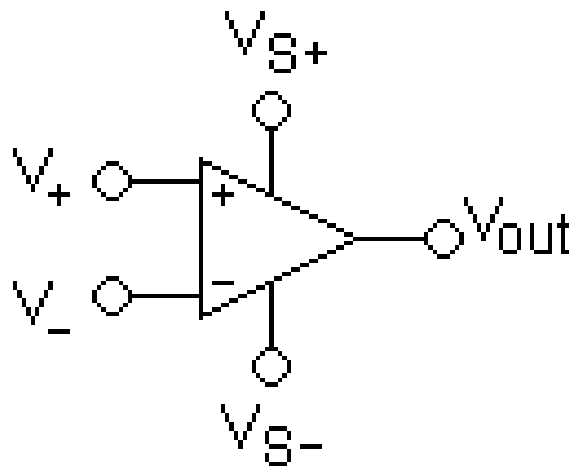


Figura 2.27 Pinout de un Amplificador Operacional.

De la Figura 2.27 se tiene: V_+ (entrada no inversora), V_- (entrada inversora), V_{OUT} (salida), V_{S+} (alimentación positiva) y V_{S-} (alimentación negativa).

Cuando se diseña un amplificador operacional, el ingeniero de circuitos integrados trabaja muy duro para conseguir que el dispositivo tenga características casi ideales. En la práctica, se encuentra que la mayoría de los amplificadores operacionales se desempeñan tan bien que, muchas veces, es posible suponer que se trabaja con un dispositivo "ideal". Las características de un amplificador operacional ideal constituyen la base de dos reglas fundamentales que quizás parezcan un poco inusuales:

1. Nunca fluye corriente hacia ninguna terminal de entrada.
2. No existe diferencia de tensión entre las dos terminales de entrada.

En un amplificador operacional real, una pequeña corriente de fuga circulará hacia la entrada, algunas veces tan baja como 40fA (femtoamperes). Es posible obtener una tensión muy pequeña entre las dos terminales de entrada; sin embargo, en comparación con las tensiones y corrientes de la mayoría de los circuitos, tales valores son tan pequeños que si se incluyeran en el análisis difícilmente afectarían los cálculos.

Al analizar circuitos de amplificadores operacionales se debe tomar otro aspecto en cuenta. En oposición a los circuitos que se han estudiado hasta ahora, un circuito de amplificadores operacionales siempre tiene una salida que depende de algún tipo de entrada. Por lo tanto, se analizarán los circuitos de amplificadores operacionales con el objetivo de obtener la expresión de la salida en términos de las cantidades de entrada. Se podrá ver que suele ser una buena idea empezar el análisis de un circuito de amplificadores operacionales en la entrada, y proceder a partir de ahí, [12].

2.5.3 DESARROLLO DEL CIRCUITO DE ACONDICIONAMIENTO

El circuito propuesto para cada canal se puede observar en la Figura 2.28 y consta de un transformador (voltaje o corriente según sea el caso), divisores de voltaje realizados con potenciómetros, un amplificador operacional que realiza operaciones aritméticas

(multiplicación y suma) y un amplificador operacional en configuración de seguidor para acoplamiento de impedancias.

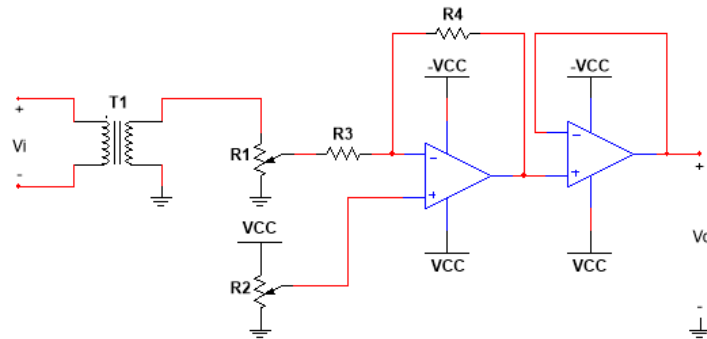


Figura 2.28 Circuito genérico propuesto para cada canal.

Lo que cambia dependiendo de la medición son las resistencias que se ocupan para la amplificación, haciendo que las 3 señales de voltaje medidas (Fase A, B y C) se reduzcan para poder ser leídas por el ADC ya que la relación de transformación de los transformadores de tensión es 10 a 1 y al leer voltajes de 127 V o 220 V sigue siendo muy alta la tensión, la reducción depende del nivel de tensión a la que se haga la medición (127 V o 220 V) ya que con un selector se escoge la resistencia que haga el factor de reducción. Para la corriente no se desea reducir la tensión aún más de lo que lo hace el transformador de corriente ya que el voltaje de salida de este es demasiado bajo y por el contrario que con los transformadores de voltaje que aun así dejan el voltaje alto para poder ser leído por el ADC, el transformador de corriente lo deja muy bajo para poder ser leído por el ADC.

Para alimentar los operacionales se diseñó un circuito en el cual con ayuda de un transformador con tap central en el secundario. Se conectan los extremos a un puente rectificador y el tap central se toma como tierra, para así crear dos voltajes, un positivo y otro negativo, ambos mayores a 12 V, con el fin de poder regularlos con un regulador de tensión positiva y un regulador de tensión negativa. El circuito propuesto para los cuatro canales (tres de voltaje y uno de corriente) es el que se muestra en la Figura 2.29.

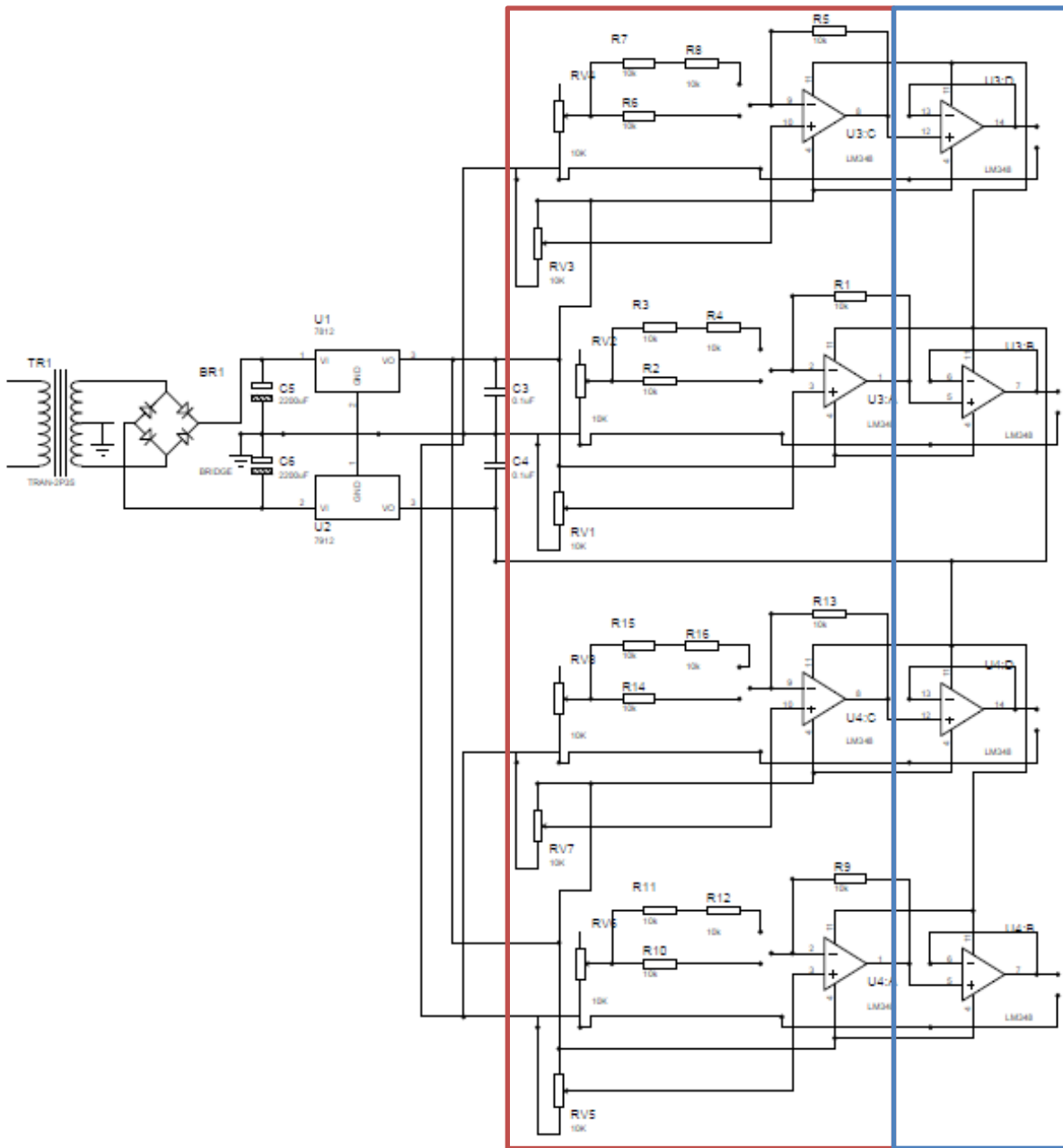


Figura 2.29 Circuito de acondicionamiento de señales para cuatro canales.

Para analizar su comportamiento en base a sus componentes, se utilizó el modelo del amplificador operacional ideal, siguiendo sus dos principales reglas (la tensión entre las entradas debe ser 0 V y la corriente entre ellas debe ser 0 A).

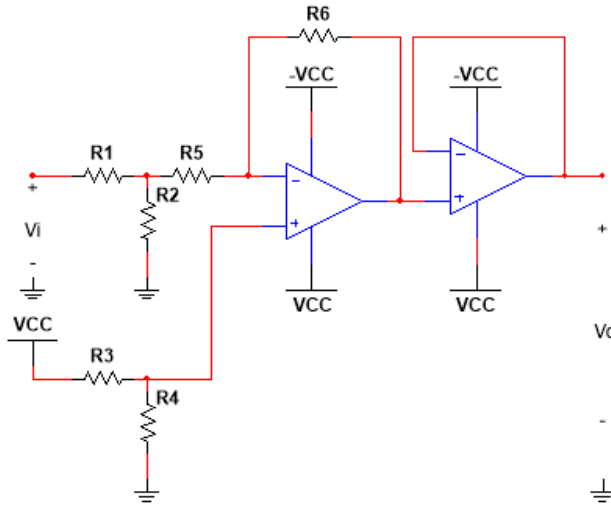


Figura 2.30 Circuito genérico para cada canal sin transformador.

Para analizar el circuito primero se toma en cuenta el circuito genérico sin transformador que es el que se muestra en la Figura 2.30 y luego se parte el circuito en dos etapas: primer amplificador operacional y segundo amplificador operacional. En la Figura 2.29 la primera etapa está delimitada en rojo, mientras que la segunda etapa se encuentra delimitada en azul.

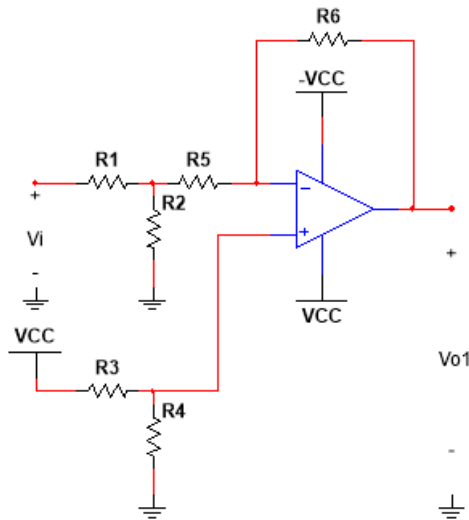


Figura 2.31 Primera etapa del circuito.

La primera etapa puede apreciarse en la Figura 2.31 y de ella se puede obtener un circuito equivalente que cumpla las leyes del amplificador operacional ideal, el cual puede apreciarse en la Figura 2.32.

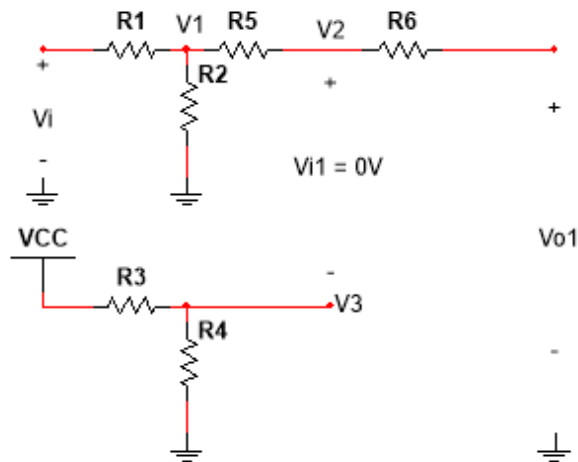


Figura 2.32 Circuito equivalente de la primera etapa del circuito.

Si se aplica análisis nodal al circuito equivalente de la primera etapa del circuito se obtienen las siguientes ecuaciones:

Del nodo de V_1 :

$$\frac{V_i - V_1}{R_1} = \frac{V_1}{R_2} + \frac{V_1 - V_2}{R_5} \quad (36)$$

Del nodo de V_2 :

$$\frac{V_1 - V_2}{R_5} = \frac{V_2 - V_{o1}}{R_6} \quad (37)$$

Del nodo de V_3 :

$$\frac{V_{CC} - V_3}{R_3} = \frac{V_3}{R_4} \quad (38)$$

Se tienen 3 ecuaciones del análisis nodal, sin embargo, si se considera que $R_1, R_2, R_3, R_4, R_5, R_6, V_{CC}$ y V_i son parámetros ya establecidos, se encuentra con un

problema ya que se tiene un sistema de ecuaciones con 3 ecuaciones y 4 variables (V_1, V_2, V_3 y V_{o1}) pero esto se puede solucionar obteniendo una ecuación en términos del voltaje entre las entradas del amplificador operacional la cual sería:

$$V_{i1} = V_2 - V_3 = 0 \quad (39)$$

De la cual se puede obtener la siguiente deducción:

$$V_2 = V_3 \quad (40)$$

Si V_2 se reemplaza por V_3 en la ecuación 38 se obtiene la siguiente ecuación:

$$\frac{V_{CC} - V_2}{R_3} = \frac{V_2}{R_4} \quad (41)$$

Con la ecuación 41 ya se tendrá un sistema de ecuaciones lineales de 3 ecuaciones y 3 variables el cual reorganizado queda de la siguiente forma:

$$\left(\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_5}\right)V_1 - \frac{1}{R_5}V_2 = \frac{V_i}{R_1} \quad (42)$$

$$\frac{1}{R_5}V_1 - \left(\frac{1}{R_5} + \frac{1}{R_6}\right)V_2 + \frac{1}{R_5}V_{o1} = 0 \quad (43)$$

$$\left(\frac{1}{R_3} + \frac{1}{R_4}\right)V_2 = \frac{V_{CC}}{R_3} \quad (44)$$

Al resolver el sistema de ecuaciones por el método de la matriz inversa se tiene:

$$\begin{pmatrix} \left(\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_5}\right) & -\frac{1}{R_5} & 0 \\ \frac{1}{R_5} & -\left(\frac{1}{R_5} + \frac{1}{R_6}\right) & \frac{1}{R_5} \\ 0 & \left(\frac{1}{R_3} + \frac{1}{R_4}\right) & 0 \end{pmatrix} \begin{pmatrix} V_1 \\ V_2 \\ V_{o1} \end{pmatrix} = \begin{pmatrix} \frac{V_i}{R_1} \\ 0 \\ \frac{V_{CC}}{R_3} \end{pmatrix} \quad (45)$$

$$\begin{pmatrix} V_1 \\ V_2 \\ V_{O1} \end{pmatrix} = \begin{pmatrix} \left(\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_5}\right) & -\frac{1}{R_5} & 0 \\ \frac{1}{R_5} & -\left(\frac{1}{R_5} + \frac{1}{R_6}\right) & \frac{1}{R_5} \\ 0 & \left(\frac{1}{R_3} + \frac{1}{R_4}\right) & 0 \end{pmatrix}^{-1} \begin{pmatrix} \frac{V_i}{R_1} \\ 0 \\ \frac{V_{CC}}{R_3} \end{pmatrix} \quad (46)$$

$$\begin{pmatrix} V_1 \\ V_2 \\ V_{O1} \end{pmatrix} = \begin{pmatrix} \frac{R_1 R_2 R_5}{R_1 R_2 + R_1 R_5 + R_2 R_5} & 0 & \frac{R_1 R_2 R_3 R_4}{(R_3 + R_4)(R_1 R_2 + R_1 R_5 + R_2 R_5)} \\ 0 & 0 & \frac{R_3 R_4}{R_3 + R_4} \\ -\frac{R_1 R_2 R_6}{R_1 R_2 + R_1 R_5 + R_2 R_5} & R_6 & \frac{(R_3 R_4)(R_1 R_2 + R_1 R_5 + R_1 R_6 + R_2 R_5 + R_2 R_6)}{(R_3 + R_4)(R_1 R_2 + R_1 R_5 + R_2 R_5)} \end{pmatrix} \begin{pmatrix} \frac{V_i}{R_1} \\ 0 \\ \frac{V_{CC}}{R_3} \end{pmatrix} \quad (47)$$

$$\begin{pmatrix} V_1 \\ V_2 \\ V_{O1} \end{pmatrix} = \begin{pmatrix} \frac{R_2 R_5 V_i}{R_1 R_2 + R_1 R_5 + R_2 R_5} + \frac{R_1 R_2 R_4 V_{CC}}{(R_3 + R_4)(R_1 R_2 + R_1 R_5 + R_2 R_5)} \\ \frac{R_4 V_{CC}}{R_3 + R_4} \\ \frac{(R_4 V_{CC})(R_1 R_2 + R_1 R_5 + R_1 R_6 + R_2 R_5 + R_2 R_6)}{(R_3 + R_4)(R_1 R_2 + R_1 R_5 + R_2 R_5)} - \frac{R_2 R_6 V_i}{R_1 R_2 + R_1 R_5 + R_2 R_5} \end{pmatrix} \quad (48)$$

De la solución anterior se obtiene una ecuación del voltaje de salida de la primera etapa del circuito con respecto al voltaje de entrada, la cual es:

$$V_{O1} = \frac{(R_4 V_{CC})(R_1 R_2 + R_1 R_5 + R_1 R_6 + R_2 R_5 + R_2 R_6)}{(R_3 + R_4)(R_1 R_2 + R_1 R_5 + R_2 R_5)} - \frac{R_2 R_6 V_i}{R_1 R_2 + R_1 R_5 + R_2 R_5} \quad (49)$$

La segunda etapa del circuito se puede apreciar en la Figura 2.33 y es un buffer que funciona para acoplar la impedancia del ADC con la del sistema, de modo que las operaciones anteriores no sean movidas por la resistencia interna del ADC.

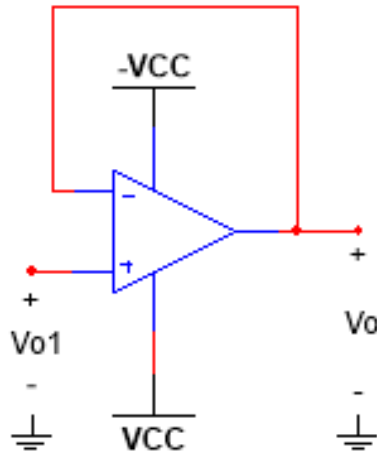


Figura 2.33 Segunda etapa del circuito.

La segunda etapa del circuito también puede ser representada en un circuito equivalente aplicando las leyes de los amplificadores operacionales ideales y el resultado es el que se muestra en la Figura 2.34.

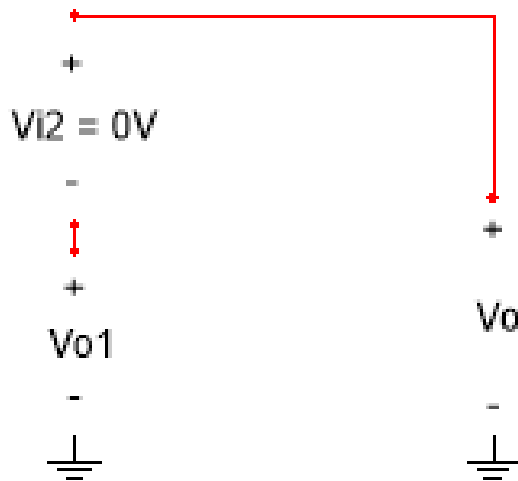


Figura 2.34 Circuito equivalente de la segunda etapa del circuito.

Del circuito anterior se puede obtener una ecuación en función de V_{i_2} la cual sería:

$$V_{i_2} = V_{o_1} - V_o = 0 \quad (50)$$

De lo cual se puede deducir fácilmente que:

$$V_0 = V_{01} \quad (51)$$

En este circuito el voltaje obtenido en la primera etapa pasa como voltaje de salida final pero aislando los circuitos por medio de una alta impedancia.

El voltaje de salida en base al voltaje de entrada se puede expresar de la siguiente forma:

$$V_0 = \frac{(R_4 V_{CC})(R_1 R_2 + R_1 R_5 + R_1 R_6 + R_2 R_5 + R_2 R_6)}{(R_3 + R_4)(R_1 R_2 + R_1 R_5 + R_2 R_5)} - \frac{R_2 R_6 V_i}{R_1 R_2 + R_1 R_5 + R_2 R_5} \quad (52)$$

Para todos los casos se cuenta con que V_{CC} es 12V por lo que la ecuación 52 quedaría:

$$V_0 = \frac{(12R_4)(R_1 R_2 + R_1 R_5 + R_1 R_6 + R_2 R_5 + R_2 R_6)}{(R_3 + R_4)(R_1 R_2 + R_1 R_5 + R_2 R_5)} - \frac{R_2 R_6 V_i}{R_1 R_2 + R_1 R_5 + R_2 R_5} \quad (53)$$

Las resistencias R_1 y R_2 al igual que R_3 y R_4 forman un potenciómetro de 10k Ω utilizado para calibrar la señal de salida según se desee, por lo que pueden ser modelados mediante las siguientes ecuaciones:

$$R_1 + R_2 = 10^4 \quad (54)$$

$$R_3 + R_4 = 10^4 \quad (55)$$

Despejando las ecuaciones 54 y 55 se obtiene:

$$R_1 = 10^4 - R_2 \quad (56)$$

$$R_3 = 10^4 - R_4 \quad (57)$$

Si se desea dejar en función del porcentaje del potenciómetro, se tiene que realizar las siguientes igualaciones:

$$\frac{R_{\%1}}{100\%} = \frac{R_2}{10k\Omega} \quad (58)$$

$$\frac{R_{\%2}}{100\%} = \frac{R_4}{10k\Omega} \quad (59)$$

Despejando las ecuaciones 58 y 59, se puede obtener ecuaciones de R_1 , R_2 , R_3 y R_4 en función de $R_{\%1}$ y $R_{\%2}$:

$$R_1 = 10^4 - 100R_{\%1} \quad (60)$$

$$R_2 = 100R_{\%1} \quad (61)$$

$$R_3 = 10^4 - 100R_{\%2} \quad (62)$$

$$R_4 = 100R_{\%2} \quad (63)$$

Por lo que, la ecuación de voltaje de salida del circuito con los valores en porcentaje de los potenciómetros quedaría:

$$V_o = \frac{12R_{\%2}(-R_{\%1}^2 + 100R_{\%1} + R_5 + R_6) - R_6R_{\%1}V_i}{100(-R_{\%1}^2 + 100R_{\%1} + R_5)} \quad (64)$$

Los valores de R_5 y R_6 dependen del caso de la medición. En total se tienen 4 diferentes casos de medición para este circuito los cuales son:

- Medición de voltaje fase A, B o C a 127 V.
- Medición de voltaje fase A, B o C a 220 V.
- Medición de corriente a 127 V.
- Medición de corriente a 220 V.

Para la medición de voltaje fase A, B o C a 127 V, se tienen los siguientes valores de R_5 y R_6 :

$$R_5 = 10k\Omega \quad (65)$$

$$R_6 = 10k\Omega \quad (66)$$

Con estos parámetros la ecuación 64 para los canales del voltaje de fase A, B o C medidos a 127 V es:

$$V_{o_{V_{ABC127}}} = \frac{12R_{\%2}(-R_{\%1}^2 + 100R_{\%1} + 2 \times 10^4) - 10^4 R_{\%1} V_i}{100(-R_{\%1}^2 + 100R_{\%1} + 10^4)} \quad (67)$$

Suponiendo que se ingresa una señal de $1 V_{DC}$ el comportamiento del voltaje de salida será el que se observa en la Figura 2.35.

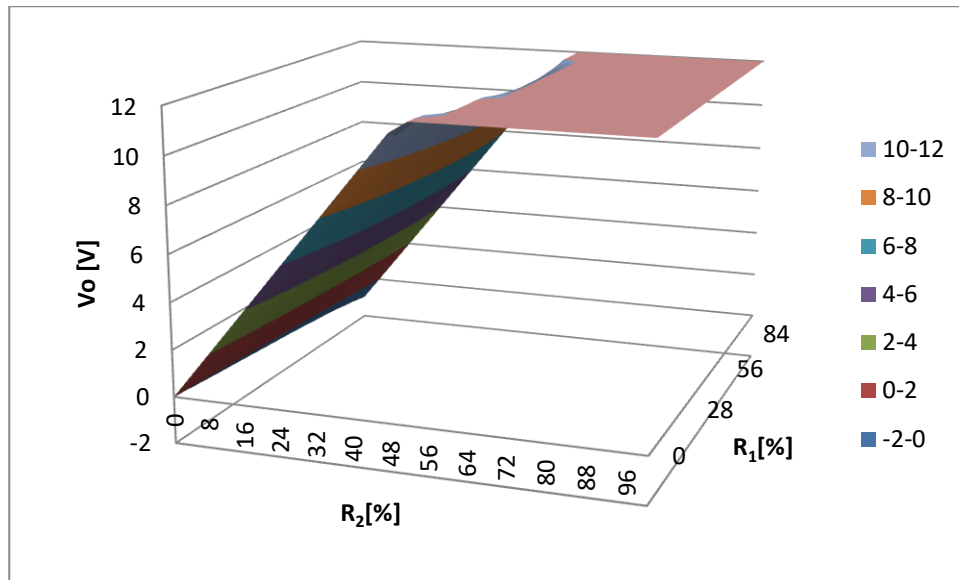


Figura 2.35 Comportamiento del circuito cuando se mide voltaje fase A, B o C en 127 V.

Al ADC se desea que le llegue una señal que se encuentre entre 0 y $4.096 V$, la señal de entrada al circuito proviene de un transformador con relación de transformación de 10 a 1, por lo que, una señal de $127 V_{RMS}$ que tiene $360 V_{pp}$, sale del transformador con un voltaje de $36 V_{pp}$ y es la señal que entra al circuito, por lo que una reducción de 10 a 1 con los amplificadores será necesaria para que el ADC pueda leerla, más un offset de $2.048 V$ para que la señal pueda ser leída por el ADC e inclusive si lleva algunos sobre voltajes también poder observarlos. Para obtener esto se tiene que observar la ecuación 67 como una ecuación lineal:

$$V_{oV_{ABC127}} = mV_i + b \quad (68)$$

donde:

$$m = \frac{-100R_{\%1}}{-R_{\%1}^2 + 100R_{\%1} + 10^4} \quad (69)$$

$$b = \frac{3R_{\%2}(-R_{\%1}^2 + 100R_{\%1} + 2 \times 10^4)}{25(-R_{\%1}^2 + 100R_{\%1} + 10^4)} \quad (70)$$

En la Figura 2.36 se puede apreciar cómo se comporta la amplificación (m) de la ecuación de voltaje dependiendo del valor en porcentaje de los dos potenciómetros, mientras que en la Figura 2.37 se aprecia el comportamiento del offset (b) dependiendo del valor de los potenciómetros en porcentaje.

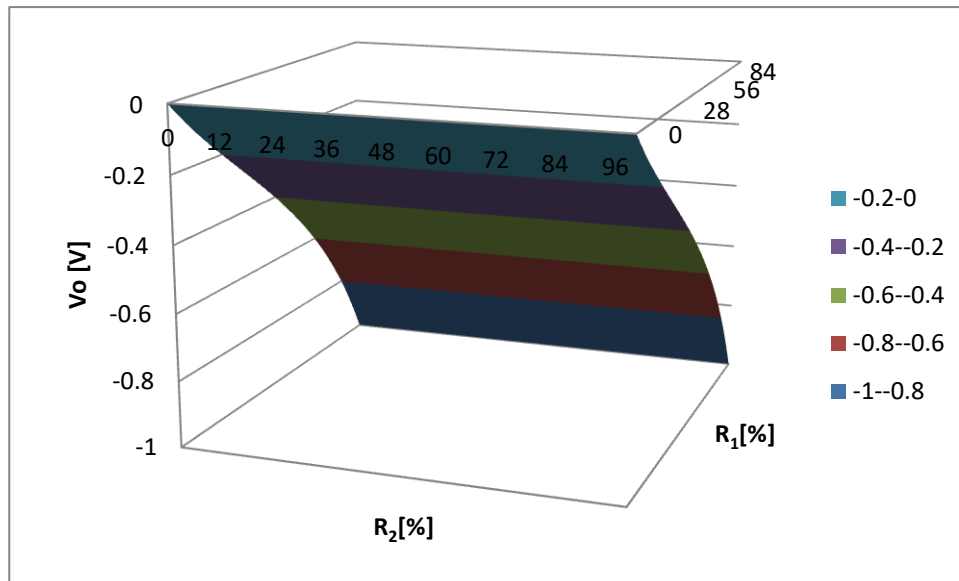


Figura 2.36 Comportamiento de la amplificación en el circuito para voltaje fases A, B o C a 127 V en base a los potenciómetros.

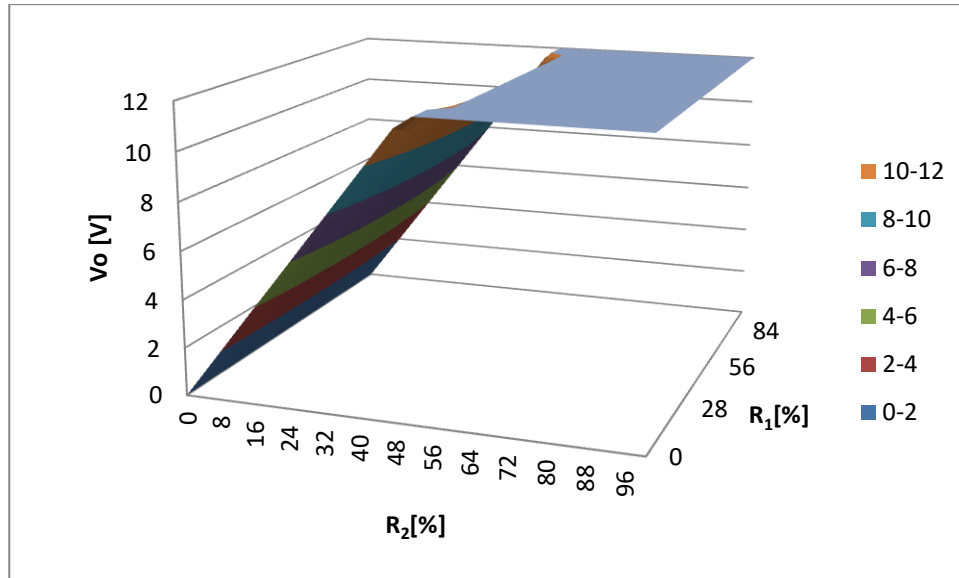


Figura 2.37 Comportamiento del offset en el circuito para voltaje fases A, B o C a 127 V en base a los potenciómetros.

Para lograr lo requerido se tiene que cumplir la siguiente condición:

$$m = -0.1 \quad (71)$$

$$b = 2.048 \quad (72)$$

Por lo que reemplazando en las ecuaciones 69 y 70 se obtiene:

$$\frac{-100R_{\%1}}{-R_{\%1}^2 + 100R_{\%1} + 10^4} = -0.1 \quad (73)$$

$$\frac{3R_{\%2}(-R_{\%1}^2 + 100R_{\%1} + 2 \times 10^4)}{25(-R_{\%1}^2 + 100R_{\%1} + 10^4)} = 2.048 \quad (74)$$

En la ecuación 73 se despeja $R_{\%1}$ y se obtiene el valor de porcentaje al que se debe ajustar el potenciómetro para que se tenga una reducción de 10 a 1 en el voltaje de entrada:

$$R_{\%1} = 10.97722286 \% \quad (75)$$

Una vez despejado se puede obtener $R_{\%2}$:

$$R_{\%2} = 8.930858234 \% \quad (76)$$

Para la medición de voltaje fase A, B o C a 220 V, se tienen los siguientes valores de R_5 y R_6 :

$$R_5 = 20k\Omega \quad (77)$$

$$R_6 = 10k\Omega \quad (78)$$

Con estos parámetros la ecuación 64 para los canales del voltaje de fase A, B o C medidos a 220 V es:

$$V_{O_{V_{ABC220}}} = \frac{12R_{\%2}(-R_{\%1}^2 + 100R_{\%1} + 3 \times 10^4) - 10^4 R_{\%1} V_i}{100(-R_{\%1}^2 + 100R_{\%1} + 2 \times 10^4)} \quad (79)$$

Suponiendo que se ingresa una señal de 1 V_{DC} el comportamiento del voltaje de salida será el que se observa en la Figura 2.38.

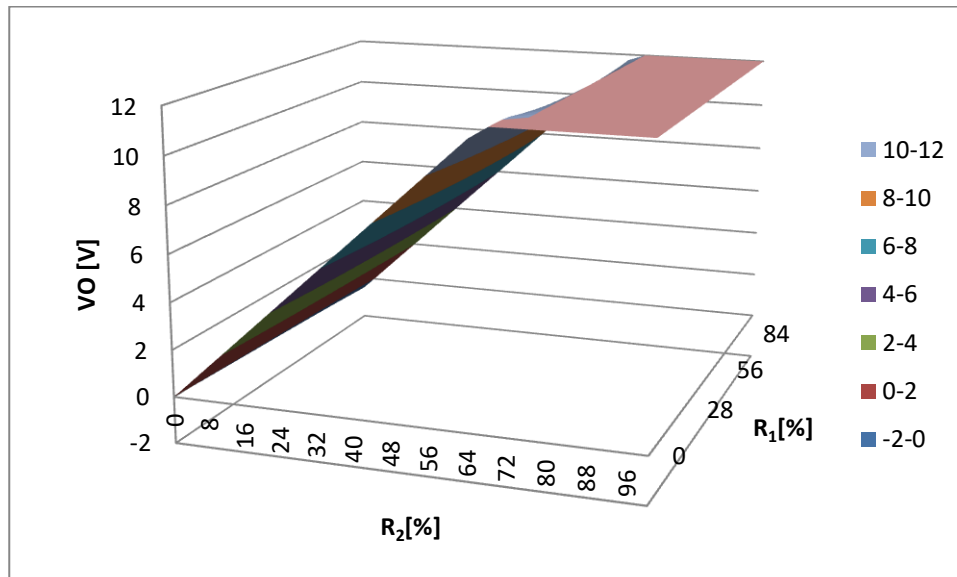


Figura 2.38 Comportamiento del circuito cuando se mide voltaje fase A, B o C en 220 V.

En el circuito físico lo que hace que cambie de 127 V a 220 V es un switch que hace que se seleccione para R_5 un valor, ya sea 10k Ω (127 V) o 20k Ω (220 V) Entonces los potenciómetros se dejarán igual que para la medición a 127 V, por lo que si se escribe la ecuación 79 como una ecuación lineal:

$$V_{oV_{ABC220}} = mV_i + b \quad (80)$$

donde:

$$m = \frac{-100R_{\%1}}{-R_{\%1}^2 + 100R_{\%1} + 2 \times 10^4} \quad (81)$$

$$b = \frac{3R_{\%2}(-R_{\%1}^2 + 100R_{\%1} + 3 \times 10^4)}{25(-R_{\%1}^2 + 100R_{\%1} + 2 \times 10^4)} \quad (82)$$

En la Figura 2.39 se puede apreciar cómo se comporta la amplificación (m) de la ecuación de voltaje dependiendo del valor en porcentaje de los dos potenciómetros, mientras que en la Figura 2.40 se aprecia el comportamiento del offset (b) dependiendo del valor de los potenciómetros en porcentaje.

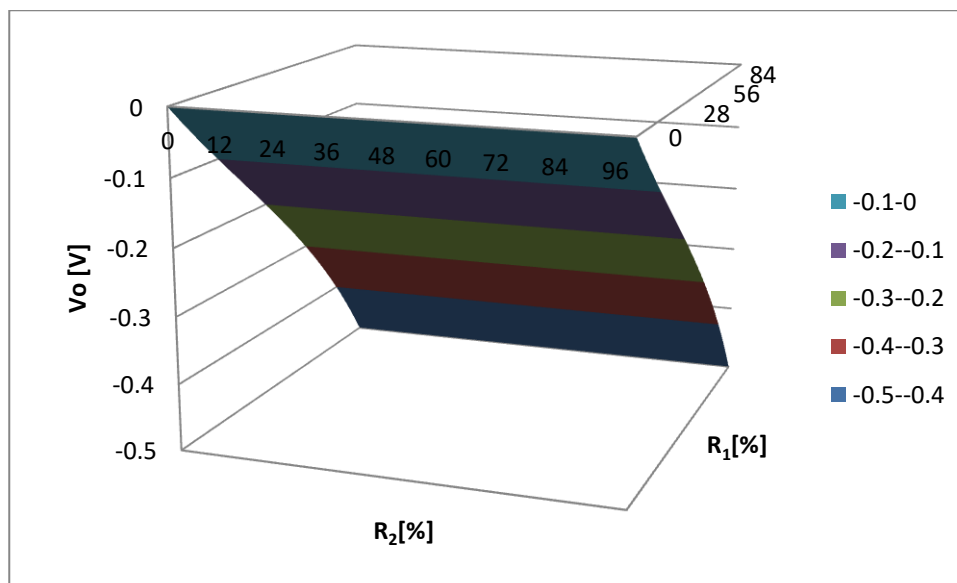


Figura 2.39 Comportamiento de la amplificación en el circuito para voltaje fases A, B o C a 220 V en base a los potenciómetros.

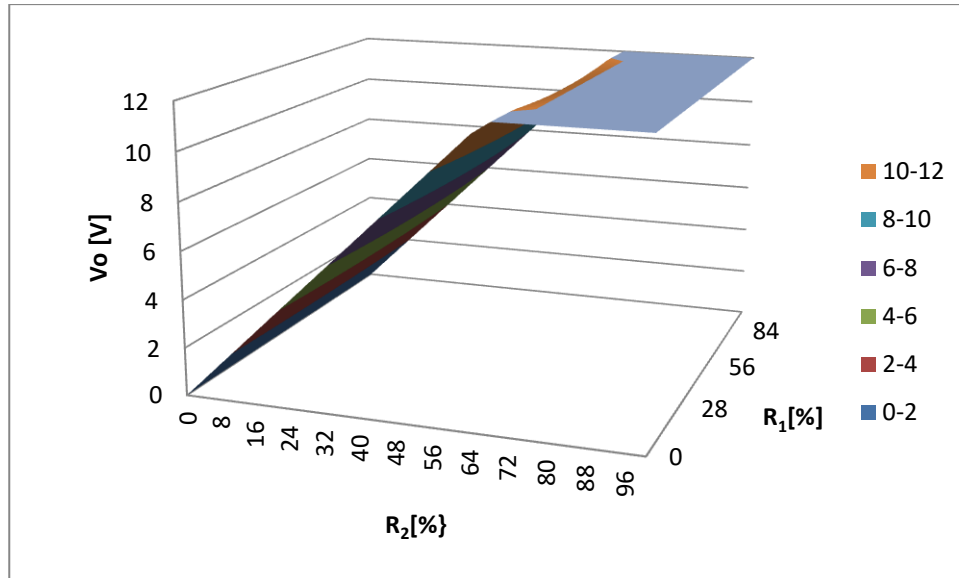


Figura 2.40 Comportamiento del offset en el circuito para voltaje fases A, B o C a 220 V en base a los potenciómetros.

El porcentaje de los potenciómetros es el mismo que para una medición de voltaje a 127 V (ecuaciones 75 y 76), por lo que reemplazando estos valores en las ecuaciones 81 y 82 se obtiene:

$$m = -0.05232924747 \quad (83)$$

$$b = 1.582591867 \quad (84)$$

Por lo que la ecuación del voltaje de salida para voltajes fases A, B o C a 220 V en función de la entrada queda de la siguiente forma:

$$V_{O_{V_{ABC220}}} = 1.582591867 - 0.05232924747V_i \quad (85)$$

Como se utiliza el mismo transformador de entrada que el de 127 V, una señal de entrada de 220 V_{rms} tiene 622 V_{pp} por lo que a la salida del transformador tendrá 62 V_{pp} y al hacer la reducción con el circuito de acondicionamiento, entraría al ADC como una señal de 3 V_{pp} con un offset de 1.58 V por lo que podría verse la señal sin problemas e inclusive algunos sobre voltajes.

Para la medición de corriente a 127 V, se tienen los siguientes valores de R₅ y R₆:

$$R_5 = 10k\Omega \quad (86)$$

$$R_6 = 100k\Omega \quad (87)$$

Con estos parámetros la ecuación 64 para los canales del corriente de medidos a 127 V es:

$$V_{O_{I127}} = \frac{12R_{\%2}(-R_{\%1}^2 + 100R_{\%1} + 11 \times 10^4) - 10^5 R_{\%1} V_i}{100(-R_{\%1}^2 + 100R_{\%1} + 10^4)} \quad (88)$$

Suponiendo que se ingresa una señal de 1 V_{DC} el comportamiento del voltaje de salida será el que se observa en la Figura 2.41.

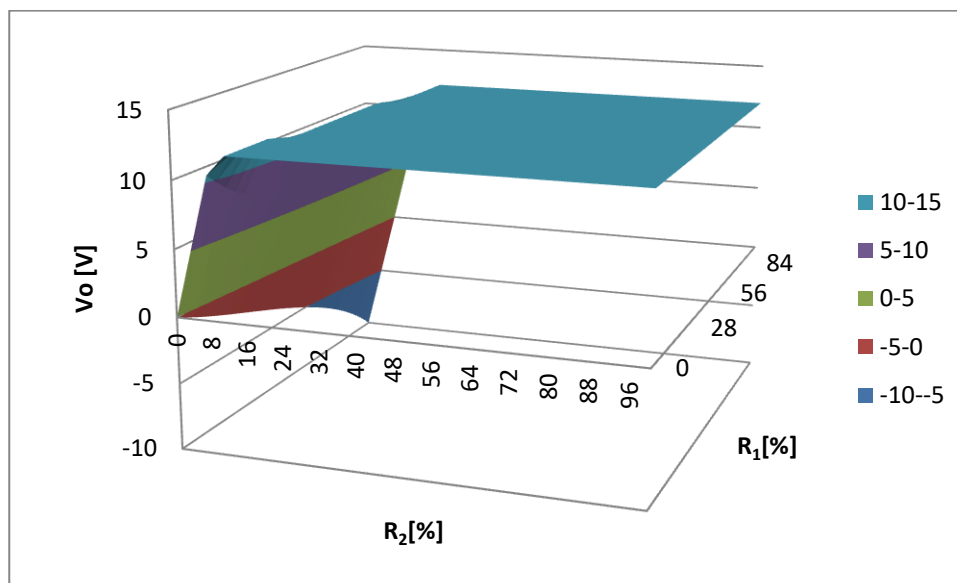


Figura 2.41 Comportamiento del circuito cuando se mide corriente en 127 V.

La señal de entrada al circuito, proviene de un transformador de corriente con relación de transformación de 100 A a 50 mA, que por su configuración el secundario a 50 mA otorga 2.78 V_{RMS}, sin embargo esos datos son solo su relación de transformación, al identificar la hoja de datos del transformador de corriente (SCT-013-000 [50]) se encontró que el diámetro máximo de un cable que puede introducirse en el transformador de corriente es de 13mm, que según la Tabla 5 de la NOM-001-SEDE 2012 [51] corresponde a un conductor aislado calibre 12 AWG con diámetro de 11.68 mm. Un conductor 12 AWG según la Tabla 310-15(b)(17) de la NOM-001-SEDE-2012

puede ser portador de una corriente de hasta 35 A_{RMS}, que con la relación de transformación anterior dará un voltaje a la entrada del circuito de 0.974 V_{RMS} que significan 2.756 V_{pp}, las señal de corriente leídas son muy pequeñas para el ADC, por lo que, se tendría que realizar una amplificación de 1 a 1.485 con los amplificadores, se le debe colocar también un offset de 2.048 V para que la señal pueda ser leída por el ADC. Para obtener esto, se tiene que observar la ecuación 88 como una ecuación lineal:

$$Vo_{I_{127}} = mVi + b \quad (89)$$

donde:

$$m = \frac{-1000R_{\%1}}{-R_{\%1}^2 + 100R_{\%1} + 10^4} \quad (90)$$

$$b = \frac{3R_{\%2}(-R_{\%1}^2 + 100R_{\%1} + 11 \times 10^4)}{25(-R_{\%1}^2 + 100R_{\%1} + 10^4)} \quad (91)$$

En la Figura 2.42 se puede apreciar cómo se comporta la amplificación (m) de la ecuación de voltaje dependiendo del valor en porcentaje de los dos potenciómetros, mientras que en la Figura 2.43 se aprecia el comportamiento del offset (b) dependiendo del valor de los potenciómetros en porcentaje.

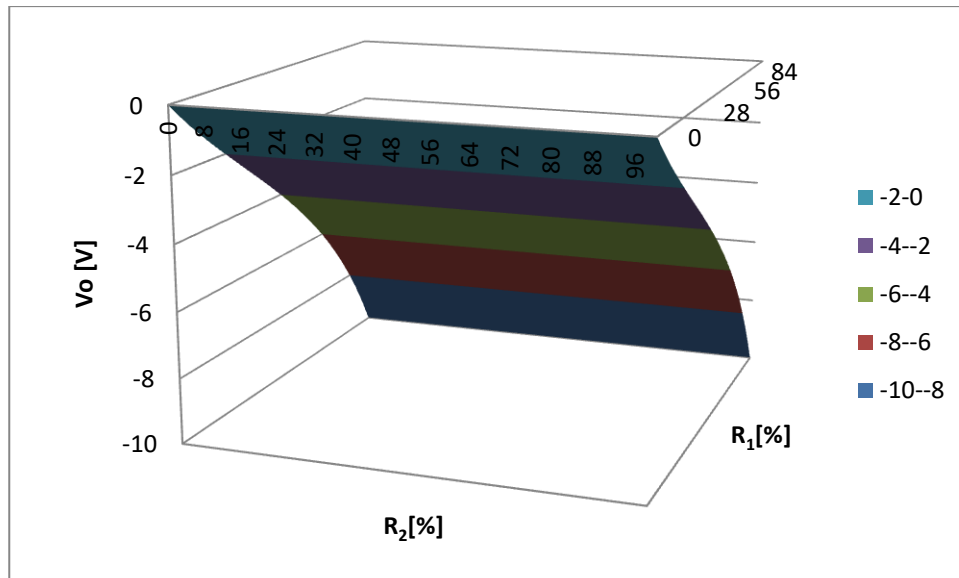


Figura 2.42 Comportamiento de la amplificación en el circuito para corriente en 127 V en base a los potenciómetros.

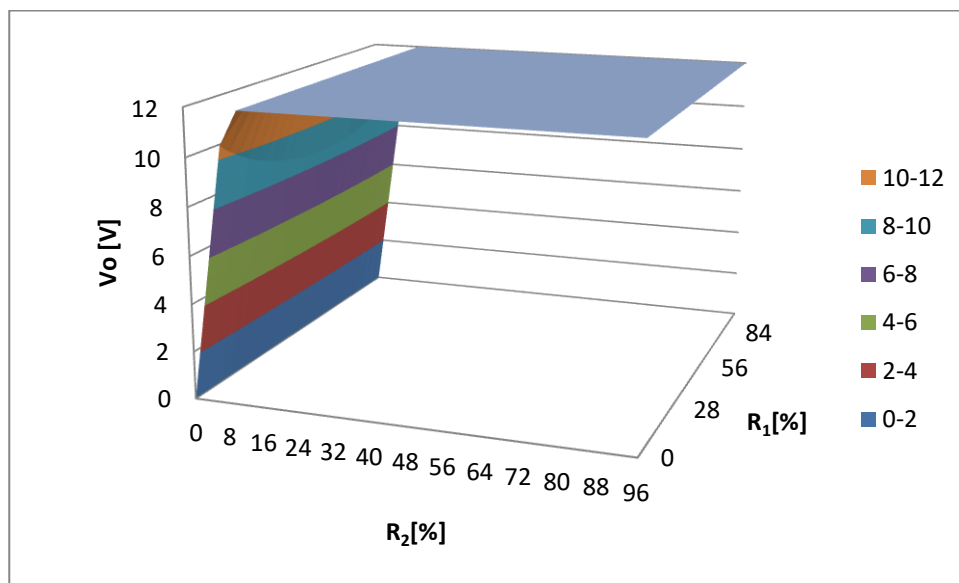


Figura 2.43 Comportamiento del offset en el circuito para corriente en 127 V en base a los potenciómetros.

Para lograr lo requerido se tiene que cumplir la siguiente condición:

$$m = -1.485984035 \quad (92)$$

$$b = 2.048 \quad (93)$$

Por lo que reemplazando en las ecuaciones 90 y 91 se obtiene:

$$\frac{-1000R_{\%1}}{-R_{\%1}^2 + 100R_{\%1} + 10^4} = -1.485984035 \quad (94)$$

$$\frac{3R_{\%2}(-R_{\%1}^2 + 100R_{\%1} + 11 \times 10^4)}{25(-R_{\%1}^2 + 100R_{\%1} + 10^4)} = 2.048 \quad (95)$$

En la 94 se despeja $R_{\%1}$ y se obtiene el valor de porcentaje al que se debe ajustar el potenciómetro para que se tenga una amplificación de 1 a 1.485 en el voltaje de entrada:

$$R_{\%1} = 16.95183677 \% \quad (96)$$

Una vez despejado se puede obtener $R_{\%2}$:

$$R_{\%2} = 1.747574314 \% \quad (97)$$

Para la medición de corriente a 220 V, se tienen los siguientes valores de R_5 y R_6 :

$$R_5 = 20k\Omega \quad (98)$$

$$R_6 = 100k\Omega \quad (99)$$

Con estos parámetros la ecuación 64 para los canales de corriente medidos a 220 V es:

$$V_{oI_{220}} = \frac{12R_{\%2}(-R_{\%1}^2 + 100R_{\%1} + 12 \times 10^4) - 10^5 R_{\%1} V_i}{100(-R_{\%1}^2 + 100R_{\%1} + 2 \times 10^4)} \quad (100)$$

Suponiendo que se ingresa una señal de 1 V_{DC} el comportamiento del voltaje de salida será el que se observa en la Figura 2.44.

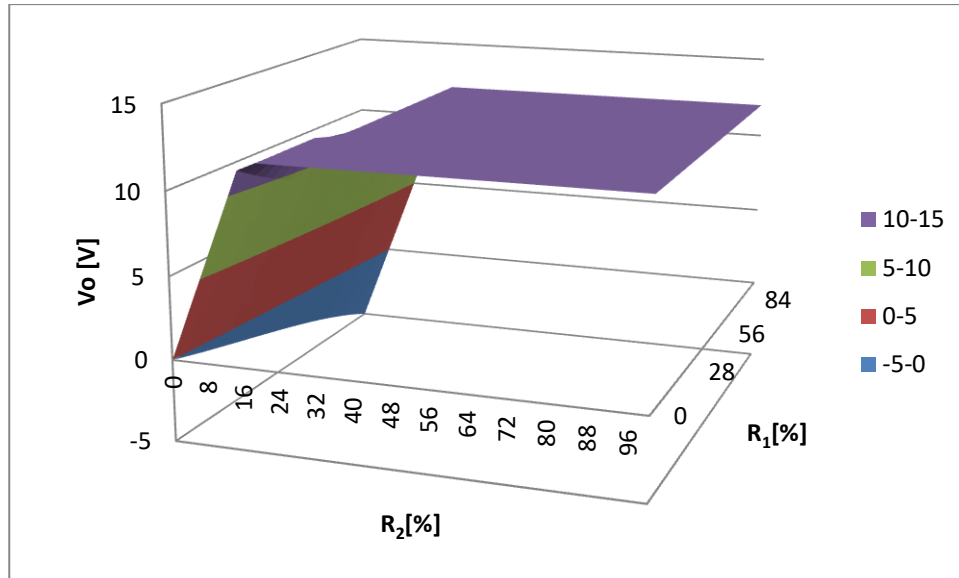


Figura 2.44 Comportamiento del circuito cuando se mide corriente en 220 V.

En el circuito físico lo que hace que cambie de 127 V a 220 V es un switch que hace que se seleccione para R_5 un valor, ya sea 10 k Ω (127 V) o 20 k Ω (220 V), entonces los potenciómetros se dejarán igual que para la medición a 127 V. por lo que si se escribe la ecuación 100 como una ecuación lineal:

$$Vo_{I_{220}} = mVi + b \quad (101)$$

donde:

$$m = \frac{-1000R_{\%1}}{-R_{\%1}^2 + 100R_{\%1} + 2 \times 10^4} \quad (102)$$

$$b = \frac{3R_{\%2}(-R_{\%1}^2 + 100R_{\%1} + 12 \times 10^4)}{25(-R_{\%1}^2 + 100R_{\%1} + 2 \times 10^4)} \quad (103)$$

En la Figura 2.45 se puede apreciar cómo se comporta la amplificación (m) de la ecuación de voltaje dependiendo del valor en porcentaje de los dos potenciómetros, mientras que en la Figura 2.46 se aprecia el comportamiento del offset (b) dependiendo del valor de los potenciómetros en porcentaje.

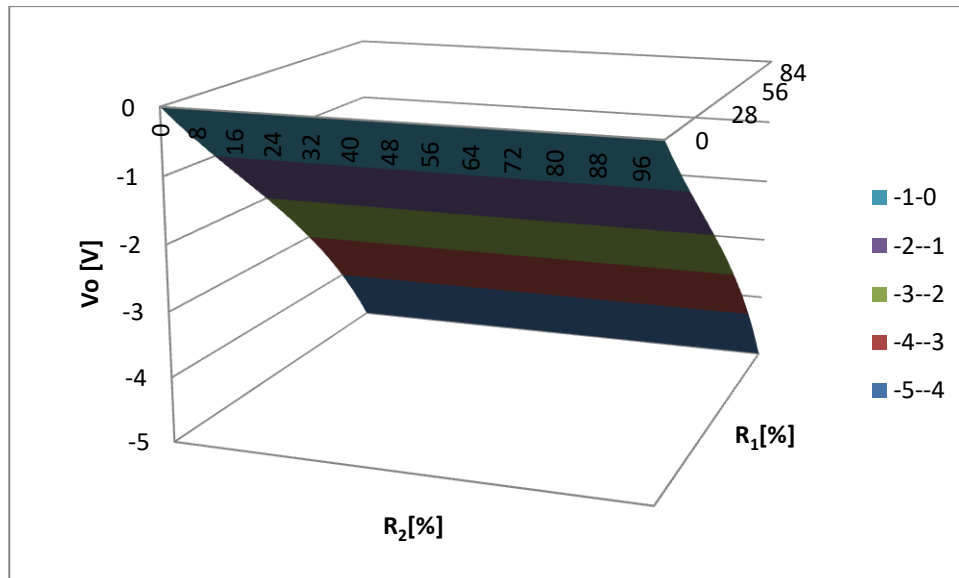


Figura 2.45 Comportamiento de la amplificación en el circuito corriente a 220 V en base a los potenciómetros.

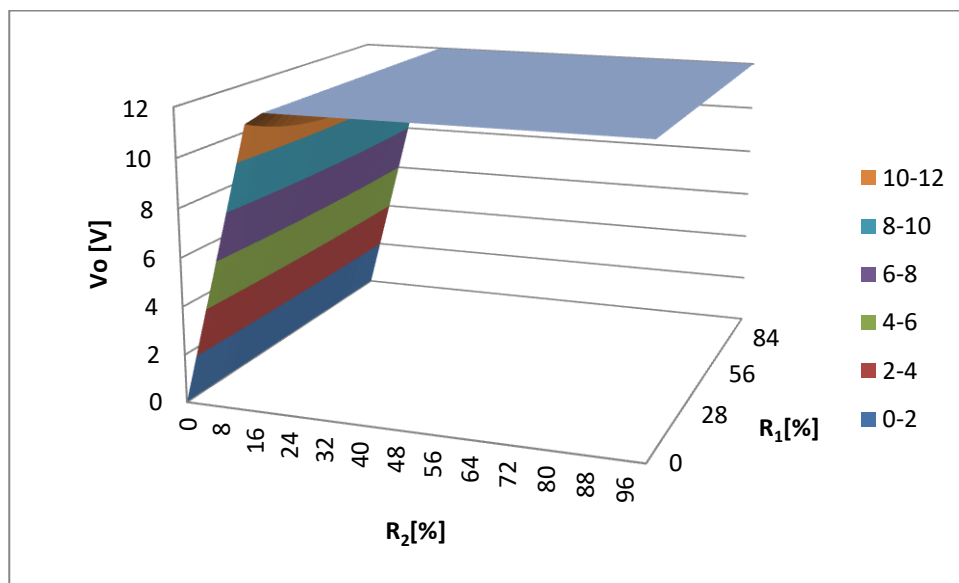


Figura 2.46 Comportamiento del offset en el circuito para corriente a 220 V en base a los potenciómetros.

El porcentaje de los potenciómetros es el mismo que para una medición de corriente a 127 V (ecuaciones 96 y 97), por lo que reemplazando estos valores en las ecuaciones 102 y 103 se obtiene:

$$m = -0.7918525864 \quad (104)$$

$$b = 1.189299219 \quad (105)$$

Por lo que, la ecuación del voltaje de salida para corrientes a 220 V en función de la entrada queda de la siguiente forma:

$$V_{o_{I_{220}}} = 1.189299219 - 0.791852864V_i \quad (106)$$

Como se utiliza el mismo transformador de entrada que el de 127 V, la corriente máxima que podrá leer el ADC a 220 V con esta configuración depende del offset ya que solo podrá leer una corriente que a la entrada del ADC tenga una amplitud de 1.189 V, al tener una amplificación de 1 a 0.791, eso significa que en el lado del secundario se tiene una corriente de 53.94 A de amplitud, lo que significan 38 A_{rms}.

Una vez analizado el circuito se debe crear un PCB con él para poder implementarlo en un gabinete junto con el FPGA. Para observar una guía rápida de cómo diseñarlo revisar el Apéndice B.

CAPÍTULO 3: ENLACE HPS-FPGA

3.1 INTRODUCCIÓN

En este capítulo se muestra como se logró el enlace entre el FPGA y el HPS de la tarjeta DE1 SoC, donde se pueda poner en operación el sistema operativo Linux Ubuntu de manera gráfica.

3.2 TECNOLOGÍAS RECONFIGURABLES

3.2.1 CPLD

Los CPLD (Complex Programmable Logic Devices) son dispositivos lógicos que son complejos y programables. Hay dos características principales de ingeniería para comprender los CPLD que los separan de los FPGA. Una característica es la arquitectura interna del dispositivo y cómo esta arquitectura implementa varias funciones lógicas. La segunda característica es la tecnología de semiconductores que

permite que los dispositivos se programen y que se conecten varias estructuras en el dispositivo.

Esencialmente, los CPLD están diseñados para aparecer como una gran cantidad de PAL (Programmable Array Logic) en un solo chip, conectados entre sí a través de un interruptor de punto de cruce. Esta arquitectura los hizo familiares para su mercado objetivo: diseñadores de placas de PC que ya estaban diseñando PAL en sus placas. Muchos CPLD se usaron para combinar varios PAL para ahorrar espacio en una placa de PC. Los CPLD utilizan las mismas herramientas de desarrollo y programación como PAL, y se basan también en las mismas tecnologías que los PAL, pero pueden manejar una lógica mucho más compleja. [14]

3.2.2 FPGA

Los FPGA reciben este nombre porque están estructurados de manera muy similar a un ASIC de matriz de puertas. Al igual que un ASIC, el FPGA consiste en una matriz regular de lógica, una arquitectura que se presta a diseños muy complejos.

Cada proveedor de FPGA tiene su propia arquitectura de FPGA, pero en términos generales todas son una variación de la arquitectura básica de un FPGA. La arquitectura básica consta de bloques lógicos configurables, bloques de entradas y salidas configurables e interconexión programable para encaminar señales entre los bloques lógicos y los bloques de entradas y salidas. Además, hay un circuito de reloj para conducir las señales de reloj a cada flip-flop en cada bloque lógico. También pueden estar disponibles recursos lógicos adicionales como ALU, memoria y decodificadores. Los dos tipos más comunes de elementos programables para un FPGA son memorias RAM estáticas.

Los CLB (Configurable Logic Blocks) contienen la lógica programable para el FPGA. Un CLB típico contiene una RAM para crear funciones lógicas combinatorias arbitrarias. También contiene flip-flops para elementos de almacenamiento de reloj y multiplexores para enrutar la lógica dentro del bloque y enrutar la lógica hacia y desde recursos

externos. Estos multiplexores también permiten la selección de polaridad, restablecer la entrada y borrar la selección de entrada.

Un bloque de entradas y salidas configurable (Configurable I/O Block) se usa para ingresar señales al chip y enviarlas a fuera nuevamente. El búfer de salida del bloque tiene controles programables para hacer que el búfer sea de tres estados o colector abierto. Estos controles permiten que el FPGA salga a la mayoría de los dispositivos TTL o CMOS estándar. El bloque contiene varios subsistemas que acondicionan las señales de entrada como de salida para evitar problemas como el ruido.

Muchas de las arquitecturas FPGA más nuevas están incorporando dispositivos complejos dentro de sus FPGA. Estos dispositivos pueden abarcar desde funciones relativamente simples, como decodificadores de direcciones o multiplicadores, hasta ALU, DSP y microcontroladores y microprocesadores. Estos dispositivos integrados están optimizados, y los dispositivos FPGA que los incluyen pueden ofrecerle una forma muy agradable de integrar un sistema completo en un solo chip, creando lo que se llama un "sistema en un chip programable" o SOPC.

La interconexión de un FPGA es muy diferente a la de un CPLD, pero es más similar a la de un ASIC de matriz de puerta. Cada CLB de un FPGA está conectado con los CLB adyacentes inmediatamente. Estas conexiones a veces se denominan líneas cortas.

Los bloques de entradas y salidas especiales con memorias intermedias especiales de reloj de alta unidad, conocidos como controladores de reloj, se distribuyen alrededor del chip. Estas memorias intermedias están conectadas a los pines de entrada del reloj y conducen las señales del reloj a las líneas de reloj globales distribuidas por todo el dispositivo en una configuración llamada árbol de reloj. Estas líneas de reloj están diseñadas para tiempos de inclinación de reloj bajos y tiempos de propagación de reloj rápidos. El diseño sincrónico es imprescindible con los FPGA, ya que los tiempos para las señales no se pueden garantizar utilizando los recursos de enrutamiento del FPGA, [14].

3.3 PROCESADORES EMPOTRADOS (ARM)

RISC (Reduced Instruction Set Computing) ocupa un lugar especial en el mapa del desarrollo de hardware, y la familia de procesadores ARM (Advanced RISC Machines) se basa en la arquitectura RISC. Gracias a configuraciones especiales, tiene la capacidad de admitir una amplia variedad de entornos. Desarrollado por especialistas británicos, ARM puede tener licencia de otras compañías que producen diferentes tipos de memoria de computadora, interfaces, radios, computadoras, dispositivos móviles, etc. Las soluciones ARM están dedicadas a las necesidades de:

- Productos de arquitectura de sistema en chips
- Productos de arquitectura de sistema en módulos.

La aparición de ARM en el mercado trajo nuevas características a los dispositivos que deben ser livianos y portátiles. Los productos ARM son las mejores soluciones para sistemas integrados como teléfonos inteligentes, tabletas o libros electrónicos. A partir de 2017, se han creado más de 100 mil millones de procesadores basados en ARM. Según esta información, las soluciones ARM son los procesadores más utilizados en el mercado, [15].

ARM ofrece a los usuarios los siguientes tipos de procesadores:

- Cortex-A: construido para sistemas operativos avanzados y exhibe el rendimiento más alto posible;
- Cortex-R: satisface perfectamente las necesidades de las aplicaciones en tiempo real y brinda a sus usuarios los tiempos de respuesta más rápidos;
- Cortex-M: construido principalmente para microcontroladores;
- SecurCore: se encarga de las aplicaciones de seguridad;
- Aprendizaje automático: para fines de aplicación de ML.

Sus capacidades y la variedad de soluciones a las que se pueden adaptar han hecho que estos dispositivos cada vez sean más utilizados.

3.4 SISTEMAS EN UN CHIP (SoC)

Un SoC es un sistema completo en un chip. Un "sistema" incluye un microprocesador, memoria y periféricos. El procesador puede ser un microprocesador personalizado o estándar, o podría ser un procesador de medios especializado para aplicaciones de sonido, módem o video. Los procesadores se interconectan mediante una variedad de mecanismos, que incluyen memorias compartidas y entidades de hardware que pasan mensajes, como redes generales en chip y canales especializados y buzones de correo. Los SoC se encuentran en todos los productos de consumo, desde módems, teléfonos móviles, reproductores de DVD, televisores, etc., [16].

3.5 SoC-FPGA

Los procesadores y los FPGA son los núcleos con más trabajo duro en la mayoría de los sistemas empotrados. La integración de la funcionalidad de gestión de alto nivel de los procesadores y las estrictas operaciones en tiempo real, procesamiento de datos extremos o funciones de interfaz de una FPGA en un solo dispositivo forma una plataforma informática integrada aún más potente que los dos por separado. Los dispositivos SoC FPGA integran arquitecturas de procesador y FPGA en un solo dispositivo. En consecuencia, proporcionan una mayor integración, menor potencia, un tamaño de placa más pequeño y una mayor comunicación de ancho de banda entre el procesador y FPGA. También incluyen un rico conjunto de periféricos, memoria en chip, una matriz lógica de estilo FPGA y transceptores de alta velocidad, [17].

En la actualidad, hay tres conjuntos de FPGA de SoC disponibles en el mercado. Los procesadores en estos dispositivos son subsistemas de procesador totalmente "dedicados" (ARM Cortex-A).

En la Figura 3.1 se puede observar gráficamente un SoC FPGA de Altera compuesto por un FPGA Cyclone V o Arria V y un procesador ARM Cortex A9. En la Figura 3.2 se muestran todos los modelos de SoC FPGA de Altera, ordenados de los más básicos a los más complejos. En la Figura 3.3 Se muestran los procesadores ARM que hay según

su tipo, están señalados el ARM Cortex A9 y el ARM Cortex A53, esto es debido a que el ARM Cortex A9 es el procesador con el que cuenta el SoC FPGA que se utilizará en este proyecto (DE1 SoC) y el ARM Cortex A53 es el procesador de más alta capacidad con el que se cuenta en estas tecnologías y es el que llevan los SoC FPGA con FPGA Stratix 10 de Altera. Cabe mencionar que un procesador ARM Cortex A53 es el que incluyen las tarjetas de desarrollo famosas Raspberry Pi.

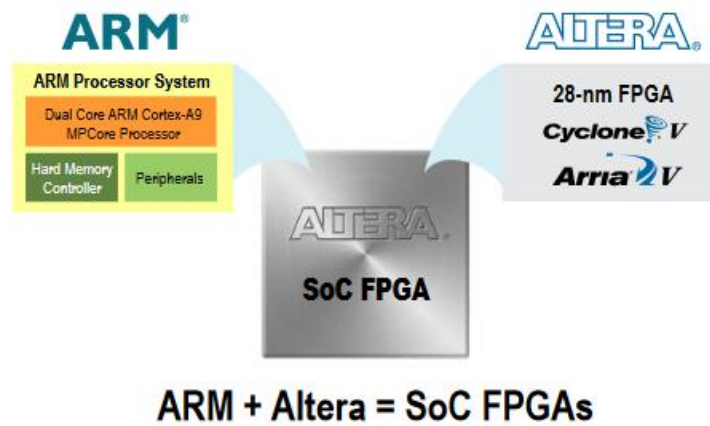


Figura 3.1 Modelo de FPGA SoC de Altera (Intel).

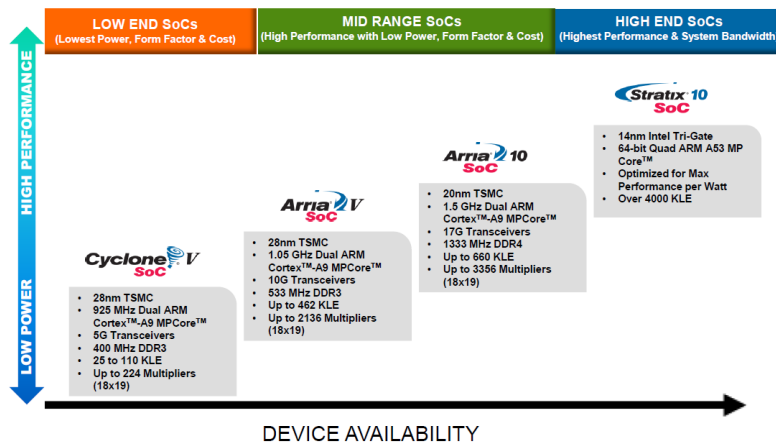


Figura 3.2 Conjuntos de SoC FPGA de Altera (Intel).

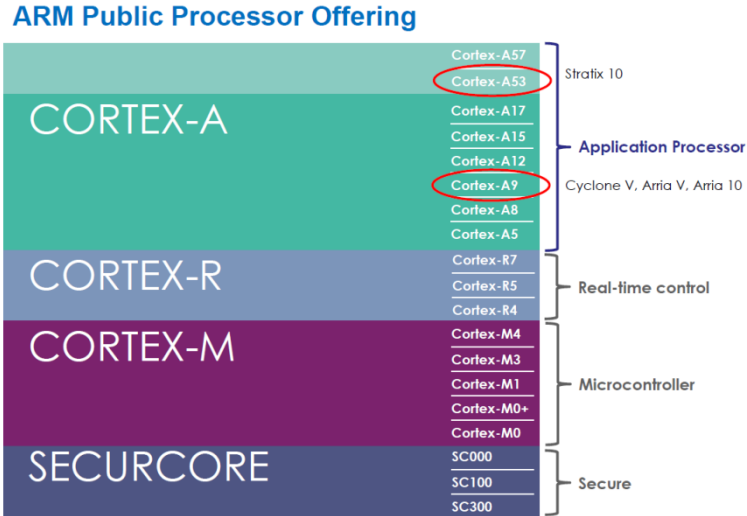


Figura 3.3 Procesadores ARM.

3.5.1 HPS-FPGA BRIDGES

El sistema en un chip (SoC) se compone de dos porciones distintas: un sistema de procesador duro ARM Cortex-A (HPS) de uno o dos núcleos y un FPGA. La arquitectura HPS integra un amplio conjunto de periféricos que reducen el tamaño de la placa y aumentan el rendimiento dentro de un sistema, [18]. En la Figura 3.4 se puede apreciar un SoC FPGA de la marca Altera (Intel) descrito en forma de bloques.

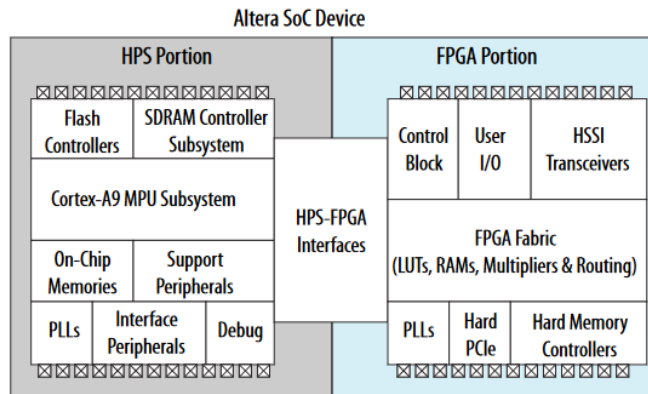


Figura 3.4 Diagrama de bloques de un dispositivo SoC de la marca Altera (Intel), [18].

Los HPS-FPGA bridges permiten al FPGA comunicarse con el HPS (Hard Processor System) y viceversa. Por ejemplo, si implementa memorias o periféricos en la estructura

FPGA, los componentes HPS pueden acceder a ellos. Los componentes implementados en el FPGA, como una arquitectura FPGA también pueden acceder a memorias y periféricos en el HPS.

El HPS contiene los siguientes HPS-FPGA bridges:

- FPGA-HPS bridge
- HPS-FPGA bridge
- Lightweight HPS-FPGA bridge

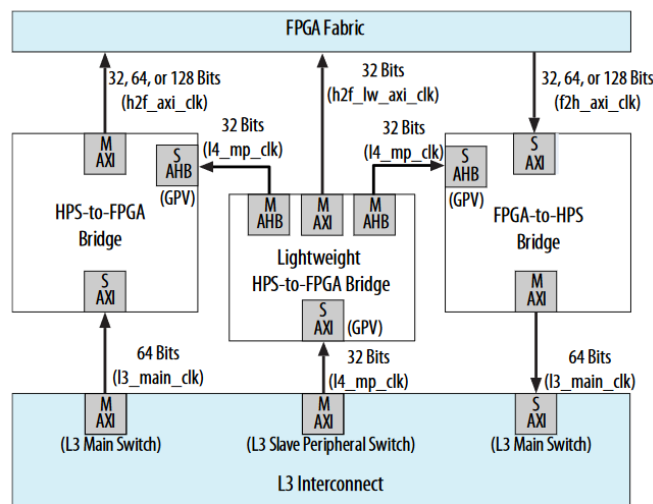


Figura 3.5 Conectividad de los HPS-FPGA Bridges, [18].

Como se muestra en la Figura 3.5 los tres bridges logran que se comuniquen el HPS y el FPGA entre sí. El FPGA-HPS bridge logra que el FPGA se comuniquen con el HPS, el HPS-FPGA bridge logra que el HPS se comuniquen con el FPGA y el Lightweight HPS-FPGA bridge logra crea una conexión entre ambos, FPGA y HPS.

En este proyecto se busca que el HPS controle al FPGA por lo que se utilizará el Lightweight HPS-FPGA. Para poder acceder al puente se hace en base a la región de memoria donde se encuentre dicho bridge. Según la hoja de datos de la tarjeta a utilizar (DE1 SoC) la dirección base es la que se muestra en la Figura 3.6.

LWHP52FPGA AXI Bridge Module Address Map

Registers in the LWHP52FPGA AXI Bridge Module.

Base Address: 0xFF400000

Figura 3.6 Dirección base para acceder al Lightweight HPS-FPGA bridge, [18].

Algo que se debe tener también en cuenta es el reloj maestro y el reset maestro para estos bridges. El reloj es proporcionado por el FPGA. El puente proporciona una lógica de cruce de reloj que permite que la lógica en el FPGA funcione en cualquier dominio de reloj, asíncrono desde el HPS. El Lightweight HPS-FPGA bridge tiene una señal de reset.

Para poder acceder al Lightweight HPS-FPGA bridge se utilizará la ayuda de la herramienta Qsys que viene instalada en el software para manejo de FPGA de Altera, Quartus. En la Figura 3.7 se puede observar el componente de Qsys que se debe colocar para poder hacer uso del HPS.

hps_0	Arria V/Cyclone V Hard Processor System			
f2h_dma_req0	Conduit	hps_0_f2h_dma_req0		
f2h_dma_req1	Conduit	hps_0_f2h_dma_req1		
f2h_dma_req2	Conduit	hps_0_f2h_dma_req2		
f2h_dma_req3	Conduit	hps_0_f2h_dma_req3		
memory	Conduit	memory		
hps_io	Conduit	hps_io		
h2f_reset	Reset Output	hps_0_h2f_reset		
h2f_axi_clock	Clock Input	Double-click to export	clk_0	
h2f_axi_master	AXI Master	Double-click to export	[h2f_axi_do...	
f2h_axi_clock	Clock Input	Double-click to export	clk_0	
f2h_axi_slave	AXI Slave	Double-click to export	[f2h_axi_do...	0x0000_0C
h2f_lw_axi_clock	Clock Input	Double-click to export	clk_0	
h2f_lw_axi_master	AXI Master	Double-click to export	[h2f_lw_axi...	
f2h_irq0	Interrupt Receiver	Double-click to export		
f2h_irq1	Interrupt Receiver	Double-click to export		

Figura 3.7 Componente de HPS en Qsys.

3.5.2 LINUX PARA SISTEMAS EMPOTRADOS

Linux debutó como Sistema Operativo de Uso General (GPOS) para la arquitectura de PC x86. Impulsado por el nacimiento de Internet, Linux permaneció fiel a sus orígenes como GPOS, en base a tres aspectos principales, los mismos que contribuyeron al porqué Linux no fue considerado inicialmente como un sistema operativo (SO) para sistemas embebidos:

- Sistema de Archivos: Linux es un SO basado en archivos, el cual requiere un sistema de archivos sobre un dispositivo de almacenamiento en bloques con derechos de lectura y escritura. Los dispositivos de almacenamiento en bloques son normalmente discos duros magnéticos giratorios, los cuales presentan una serie de componentes mecánicos, los cuales no son muy prácticos para sistemas embebidos.
- Unidad de Manejo de Memoria (MMU): Linux es un SO multitarea, el switch entre tareas requiere que los procesos individuales tengan un espacio privado de direcciones de memoria, el cual puede ser fácilmente mapeado en la memoria física, cuando el proceso está corriendo en la CPU. Los microcontroladores que han sido usados normalmente en aplicaciones embebidas, no poseen este tipo de unidad de memoria.
- Tiempo Real: Algunos sistemas embebidos que ejecutan aplicaciones críticas podrían requerir respuestas donde el tiempo es un factor importante, y Linux lamentablemente no incluye la funcionalidad tiempo-real como parte de su kernel principal.

Sin embargo, los avances en las tecnologías de los semiconductores han ayudado a superar todas estas limitaciones mencionadas anteriormente. Memorias flash confiables y de bajo costo, poderosos SoC han logrado despertar el interés de los ingenieros en reemplazar los microcontroladores por estos chips para sus soluciones embebidas [19].

Algunas versiones embebidas de distribuciones completas de Linux son:

- Debian
- Fedora
- Gentoo
- SUSE
- Ubuntu

En el caso de los SoC FPGA, cuentan con un procesador ARM Cortex-A los cuales son capaces de soportar sistemas operativos. Para los equipos SoC FPGA de Altera existen versiones que pueden descargarse de manera libre y son:

- Linux Yocto: Para manejar el SoC FPGA desde una terminal y por medio de la interfaz JTAG y una computadora. Un ejemplo de este se puede observar en la Figura 3.8 donde se puede ver el sistema operativo corriendo desde una terminal PuTTY desde un ordenador.
- Linux Ubuntu: Para manejar el SoC FPGA con un sistema operativo autónomo de manera gráfica, solo necesita conectarse un mouse, un monitor y un teclado al SoC FPGA. En la Figura 3.9 hay un ejemplo de sistema operativo gráfico LXDE que se puede utilizar en una tarjeta DE1 SoC.

```

COM7 - PuTTY
EXT2-fs (mmcblk0p2): error: couldn't mount because of unsupported optional featu
res (244)
usb 1-1: new high-speed USB device number 2 using dwc2
usb 1-1: New USB device found, idVendor=0424, idProduct=2512
usb 1-1: New USB device strings: Mfr=0, Product=0, SerialNumber=0
hub 1-1:1.0: USB hub found
hub 1-1:1.0: 2 ports detected
usb 1-1.2: new high-speed USB device number 3 using dwc2
usb 1-1.2: New USB device found, idVendor=07d1, idProduct=3a09
usb 1-1.2: New USB device strings: Mfr=16, Product=32, SerialNumber=48
usb 1-1.2: Product: 11n adapter
usb 1-1.2: Manufacturer: ATHER
usb 1-1.2: SerialNumber: 12345
random: nonblocking pool is initialized
EXT4-fs (mmcblk0p2): 2 orphan inodes deleted
EXT4-fs (mmcblk0p2): recovery complete
EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. Opts: (null)
VFS: Mounted root (ext4 filesystem) on device 179:2.
devtmpfs: mounted
Freeing unused kernel memory: 364K (806c7000 - 80722000)
init: ureadahead main process (52) terminated with status 5
Last login: Thu Jan  1 00:00:13 UTC 1970 on tty1
root@delsoclinux:~#

```

Figura 3.8 Linux Yocto en el SoC FPGA.



Figura 3.9 Linux Ubuntu en SoC FPGA.

3.6 ENLACE ENTRE FPGA Y HPS CON LINUX UBUNTU GRÁFICO

El primer paso para realizar el enlace, es tener disponible la arquitectura preparada para lanzar el Linux Ubuntu de manera gráfica [20].

Ya que se tiene la arquitectura, se debe abrir el proyecto de Quartus tal como se muestra en la Figura 3.10.

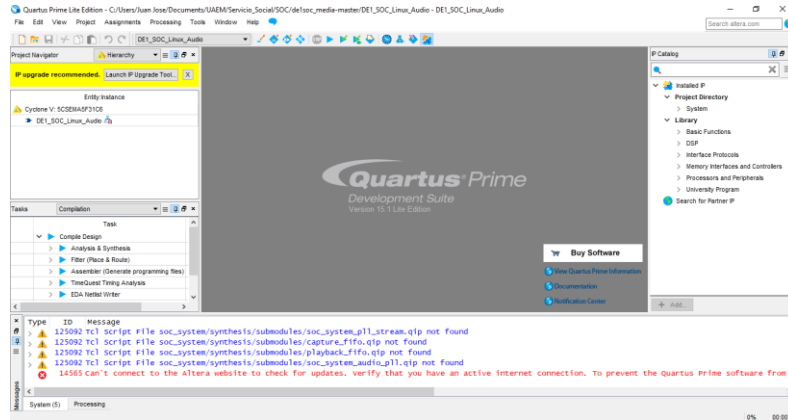


Figura 3.10 Proyecto descargado DE1_SOC_Linux_Audio.

En este caso la arquitectura que se quiere que tenga el Linux en modo gráfico, es el sistema de adquisición de señales hecho previamente, es por eso que se deben incluir los archivos VHDL al proyecto, desde la pestaña Project, en la opción Add/Remove Files in Project y ahí seleccionar los archivos VHDL del proyecto en cuestión. Los archivos que se deben añadir se muestran en la Figura 3.11.

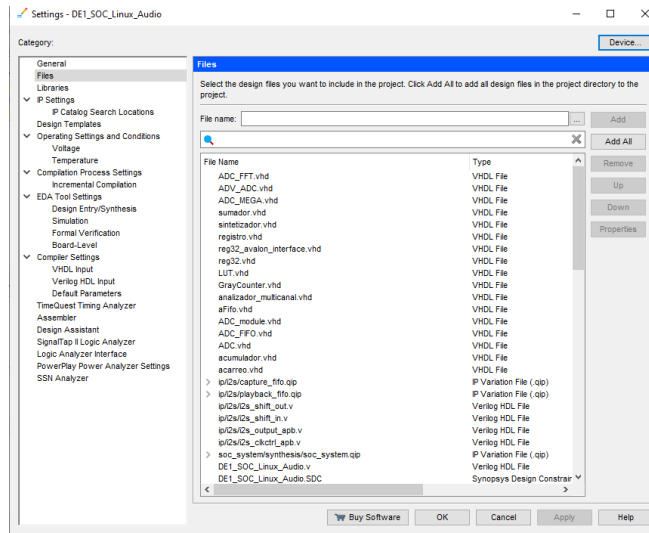


Figura 3.11 Archivos del sistema de adquisición de señales añadidos.

Luego se debe abrir Qsys y el archivo llamado soc_system.qsys, el cual tendrá la forma que se indica en la Figura 3.12.

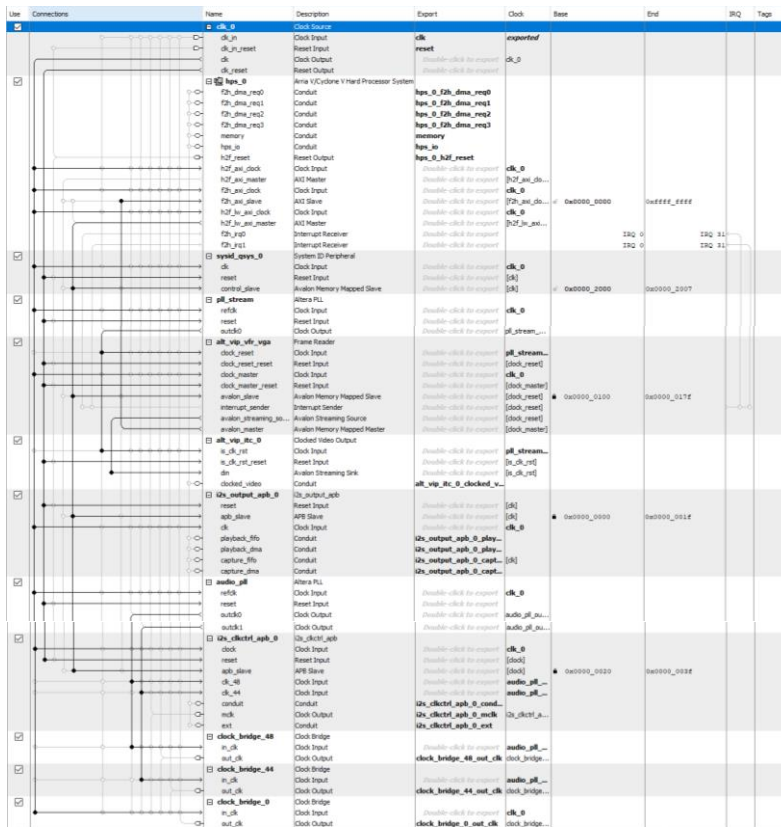


Figura 3.12 Archivo descargado: soc_system.qsys.

No se deberá quitar ni modificar lo que hay en el archivo si es que se quiere que funcione el modo gráfico. En este archivo de Qsys se pueden añadir los componentes que se requiera, esto depende directamente de las entradas y salidas que se quieran controlar o leer del sistema en cuestión. En este caso las entradas del sistema que se desean controlar son: El reloj de lectura de la FIFO de las muestras del ADC (POP), el reloj de lectura de la FIFO de las FFT (POP_FFT), la habilitación de las FIFO (enable) y la entrada de control de la frecuencia de muestreo (F_s). Las salidas que se desean leer son: Las 4 señales muestreadas por el ADC (output, output2, output3 y output4), las 4 FFT de las señales muestreadas por el ADC (output_fft, output2_fft, output3_fft y output4_fft) y la bandera que indica cuando se terminó de calcular la FFT (FFT_done).

Para saber que componentes utilizar, se utiliza la lógica que las entradas que se desean controlar deben ser una salida en el Qsys ya que el HPS le mandará la señal de control a la entrada del sistema y las salidas que se deseen leer deben ser entradas en el Qsys ya que se almacenarán en una localidad de memoria del HPS de donde se podrán leer.

Para las salidas del Qsys se utiliza el componente llamado PIO el cual se muestra en la Figura 3.13. El componente PIO se encuentra en la carpeta Processors and Peripherals y en la subcarpeta Peripherals. En su ventana de edición se le puede asignar la dirección del componente (entrada, salida, bidireccional, etc.) y se deberá seleccionar como salida; también se puede seleccionar el tamaño de bits de la salida, el cual será seleccionado según el tamaño de bits de la entrada del sistema con el que se interconectará.

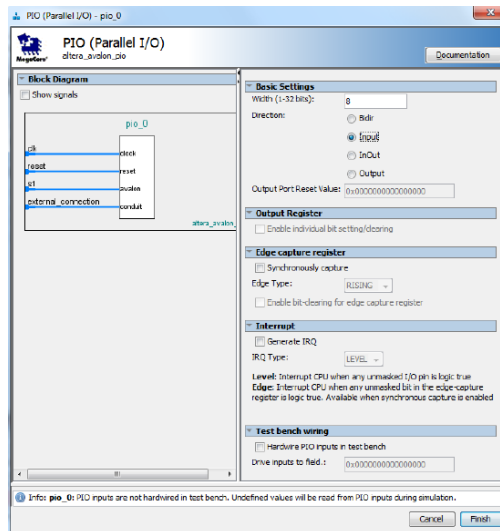


Figura 3.13 Ventana de configuración de PIO (Componente de Qsys).

Para las entradas de Qsys se utilizó un componente creado por el usuario, el cual funciona como un registro que almacena la información que entre a él para poder ser leído en una cierta localidad de memoria del HPS. Para agregar un componente, se requiere estar en la ventana de Qsys y en la parte en la que se añaden componentes se selecciona la opción llamada New. Lo cual desplegará una ventana como la que se muestra en la Figura 3.14. Las pestañas a utilizar en dicha ventana son Component type, Files, Signals e Interfaces. Si alguna de estas no aparece se puede editar desde la pestaña View. En Component Type se debe especificar el nombre del componente así como la carpeta en la que se guardara (se puede crear una nueva carpeta como es el caso del ejemplo) [21].

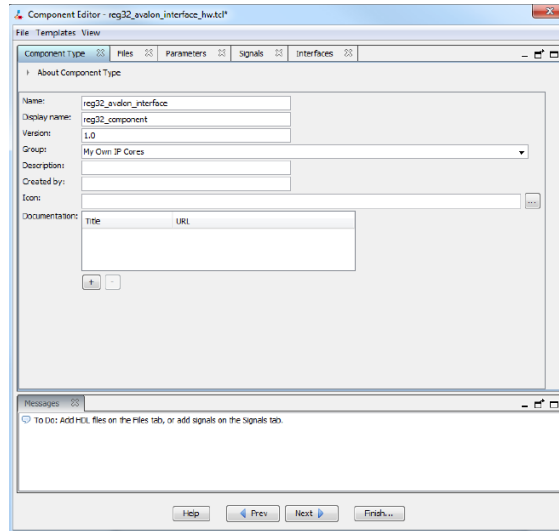


Figura 3.14 Creación de nuevo componente en Qsys.

En la sección llamada Files hay que agregar los dos archivos VHDL que hacen que el componente funcione.

Para añadir los archivos VHDL se debe oprimir el botón + y añadir los dos archivos. La ventana tendrá dos opciones para añadir archivos, se oprimirá el botón llamado Copy from Synthesis File en la parte que dice VHDL Simulation Files. Una vez hecho esto se oprimirá el botón Analyze Synthesis Files para poder pasar al siguiente paso. La ventana debe observarse como se muestra en la Figura 3.15.

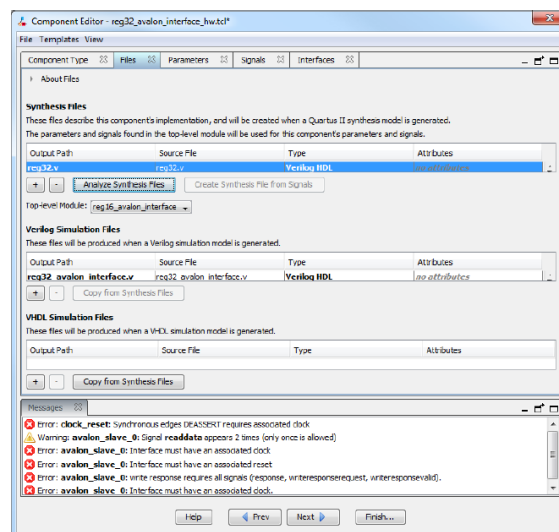


Figura 3.15 Adición de los archivos VHDL al componente creado para Qsys.

En la parte de Signals se debe asignar que tipo de interface y de señal son las entradas y salidas de la arquitectura diseñada en VHDL, para cambiar esto se debe seleccionar el renglón correspondiente y seleccionar lo adecuado, por ejemplo para el reloj se selecciona new Clock input en la interface y como tipo de señal clk; para el resetn se debe escoger new Reset input en la interface y como tipo de señal reset_n; para la salida que es Q_export se selecciona Conduit como interface. Al final las señales deben quedar de como se muestra en la Figura 3.16.

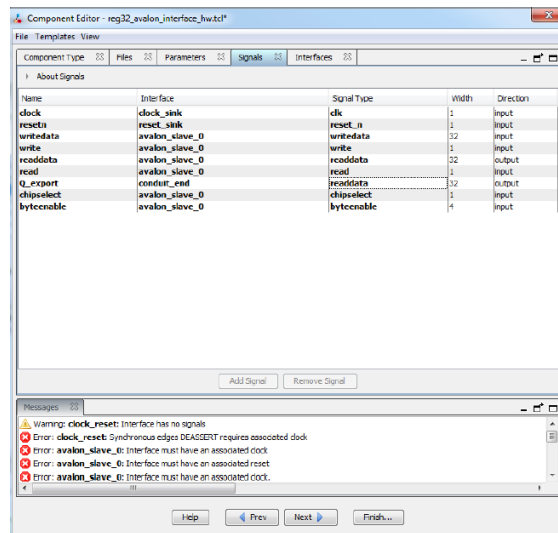


Figura 3.16 Asignación de señales para el componente creado para Qsys.

En la parte de interfaces lo que se debe hacer es asignar a todas las interfaces clock_sink como reloj asociado y reset_sink como reset asociado. También en la interface de avalon_slave_0 hay una sección que dice Timing, en este punto se debe poner el tiempo en ciclos de reloj que puede esperar para tener una respuesta, como el registro debe actuar de manera inmediata en la casilla Read wait se cambiara a 0. Los cambios que se deben hacer son como los que se muestran en la Figura 3.17.

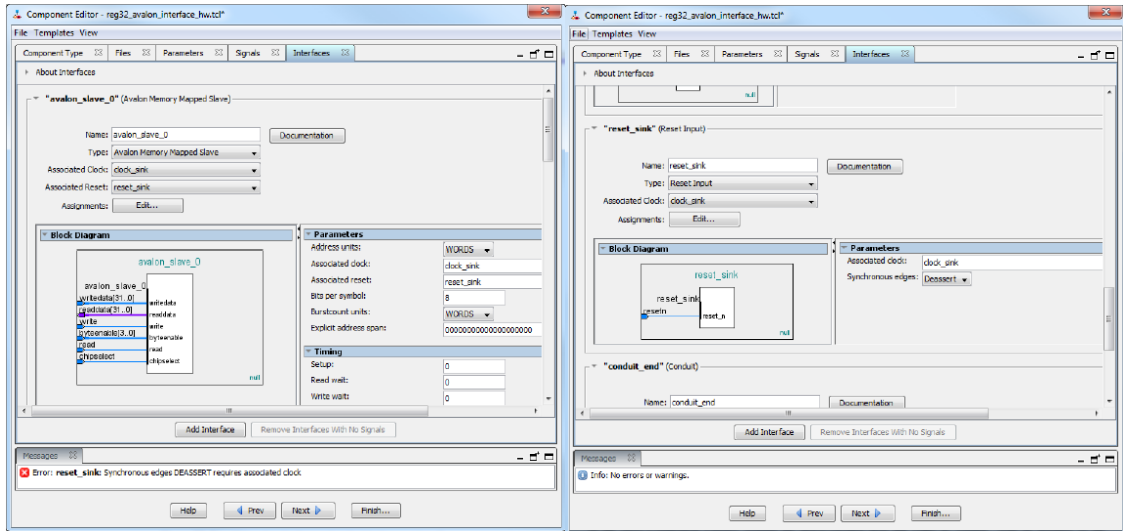


Figura 3.17 Configuración de interfaces para el componente creado para Qsys.

Una vez hecho lo anterior, se oprime en Finish y el componente está listo para ser utilizado en un sistema. Para llamarlo solo hay que buscarlo en el catálogo donde se buscan los componentes, este se encontrará en la carpeta que se creó y al darle añadir se abrirá una ventana de edición como la de cualquier otro componente, así como se muestra en la Figura 3.18, finalmente solo habrá que oprimir el botón Finish para añadir el componente creado.

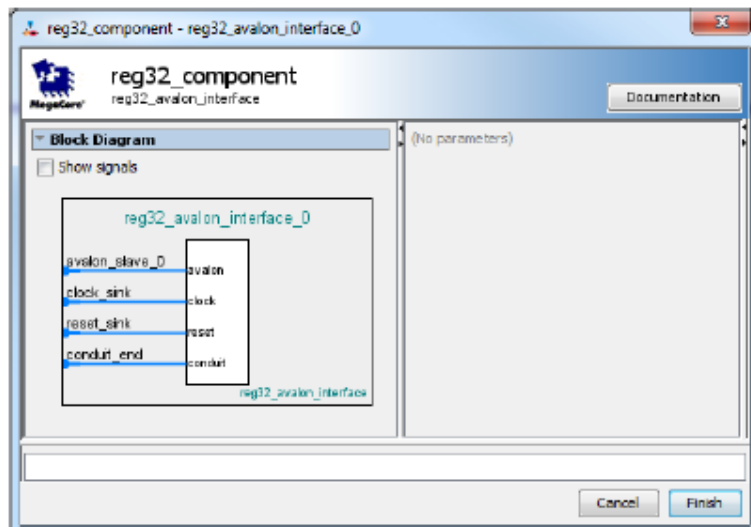


Figura 3.18 Componente creado para Qsys.

Hay que notar que las conexiones de control de los componentes añadidos en la Figura 3.19 se conectan al HPS, y con ayuda de la herramienta Assign Base Adresses se asignan las direcciones de memoria del HPS al que se conectan dichas entradas y salidas.

Ahora se debe generar el VHDL, pero antes se debe borrar el archivo de verilog existente, este se encuentra en la carpeta del proyecto, en la carpeta soc_system, en la carpeta synthesis. El archivo que debe eliminarse es el señalado en la Figura 3.20.

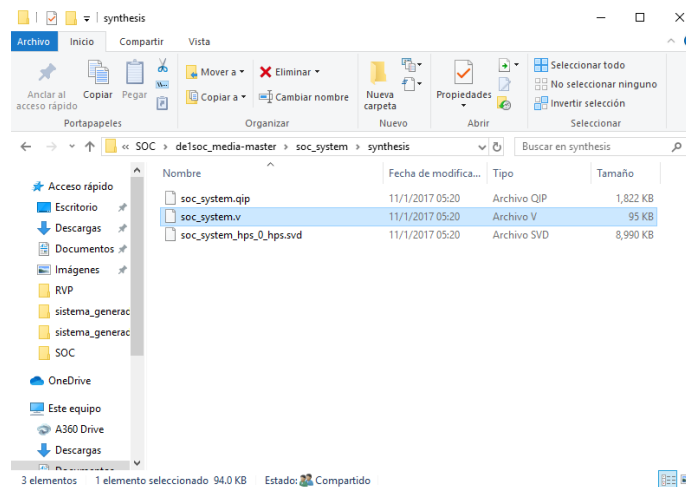


Figura 3.20 Archivo de verilog a eliminar.

Ahora hay que generar un archivo que contenga el sistema hecho en Qsys, para esto primero se guarda el sistema. Luego en la pestaña Generate, se oprime el botón Generate HDL, en este punto se puede escoger entre VHDL o Verilog como lenguaje del archivo a generar, se selecciona VHDL, y luego se oprime el botón Generate. La ventana donde se hacen las configuraciones anteriores es la que se muestra en la Figura 3.21.

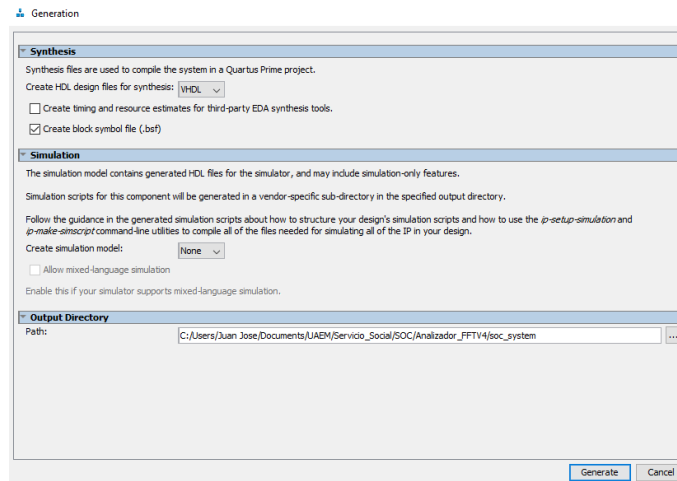


Figura 3.21 Generación de archivos VHDL del sistema realizado en Qsys.

Para poder utilizar el sistema se necesita instanciar en un código de VHDL y unirlo con el sistema de adquisición de señales, el archivo generado se encuentra en los archivos del proyecto en un archivo llamado soc_system.qip, si se extiende ese archivo desde Quartus se podrá observar el archivo generado llamado soc_system.vhd el cual tiene por entidad la mostrada en la Figura 3.22.

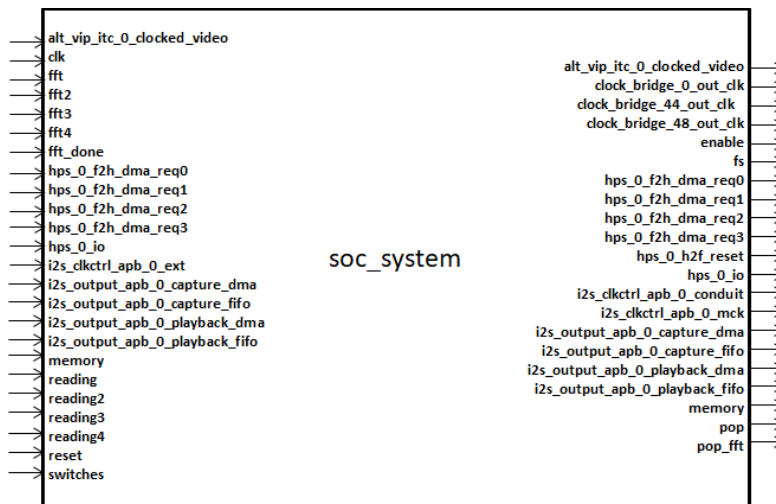


Figura 3.22 Entidad del archivo generado.

El código que se utiliza para instanciar tanto el sistema generado en Qsys como el sistema de adquisición de señales debe ser capaz también de instanciar todos los pines del FPGA en su máxima entidad, así como en el código también se deben realizar las

conexiones con la salida de VGA del FPGA. El código que será la máxima entidad en este proyecto puede ser observado en la Figura 2.23, de forma que se ven sus entradas y salidas.

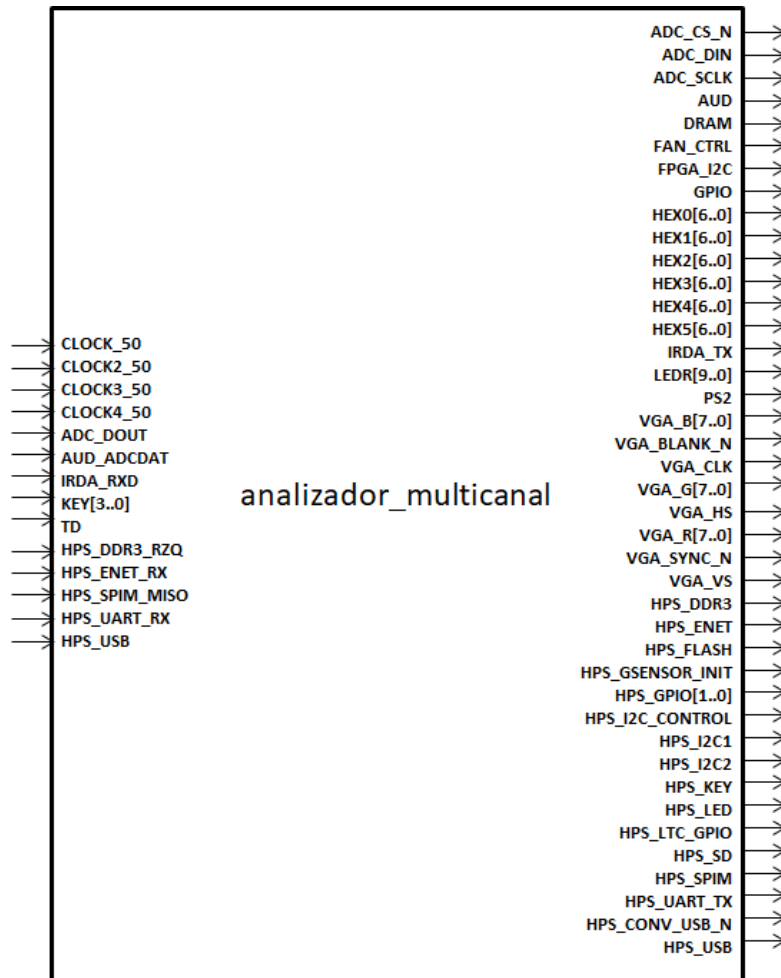


Figura 3.23 Máxima entidad del sistema analizador multicanal.

Finalmente, se debe hacer de la arquitectura descrita anteriormente la de más alto nivel, compilarlo y una vez terminado esto se deben convertir el archivo .sof generado, llamado DE1_SOC_Linux_Audio.sof a un .rbf llamado soc_system.rbf con ayuda del convertidor de archivos del Quartus. El ejemplo de cómo convertir los archivos de salida de .sof a .rbf se observa en la Figura 3.24.

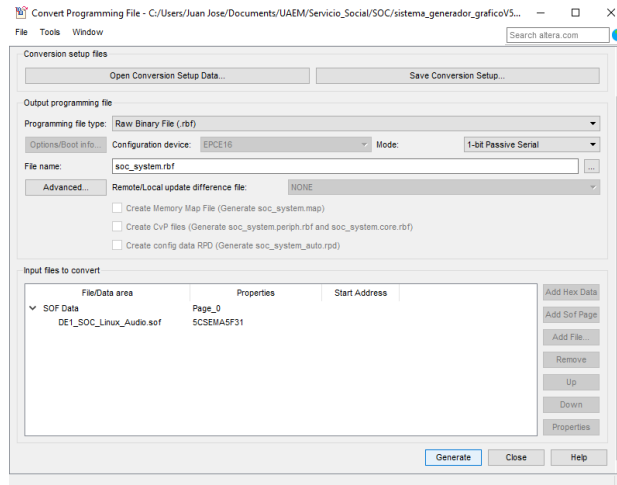


Figura 3.24 Convertidor de archivos de salida de Quartus.

El archivo generado se debe colocar en una microSD donde se tenga la imagen de Linux cargada sobre ella. (Para ver como montar Linux Ubuntu de la DE1 SoC en una microSD, ver Apéndice C.) Al abrir microSD se notará que tiene varios archivos instalados, entre ellos hay uno llamado `soc_system.rbf`, dicho archivo tiene que ser reemplazado por el generado en Quartus tal y como se muestra en la Figura 3.25.

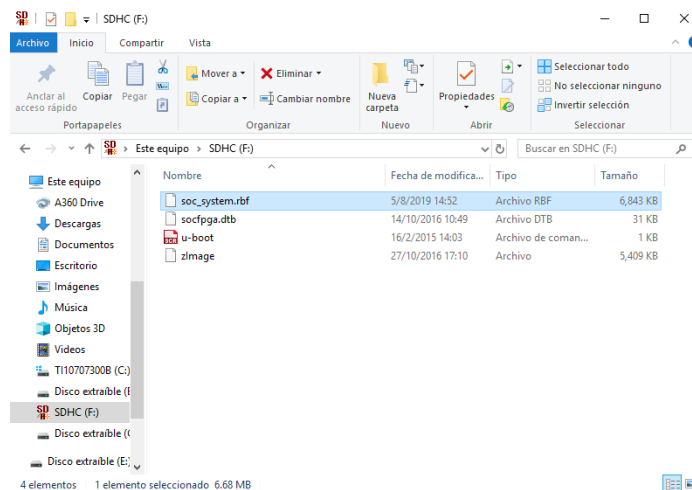


Figura 3.25 Archivo “`soc_system.rbf`” generado en Quartus montado en la microSD.

La tarjeta DE1 SoC cuenta con un selector en la parte de abajo, el cual permite según su selección escoger el modo en el que se utilizará la DE1 SoC, los modos pueden ser: FPGA solamente, FPGA con Linux terminal (controlado mediante una interfaz JTAG), Linux gráfico, etc. Para poner el FPGA EN MODO DE Linux modo gráfico se debe

poner el selector en “00000”. En la Figura 3.26 se muestra donde se encuentra el selector MSEL de la tarjeta DE1 SoC y como se deben dejar los switches del selector para que funcione con Linux gráfico.

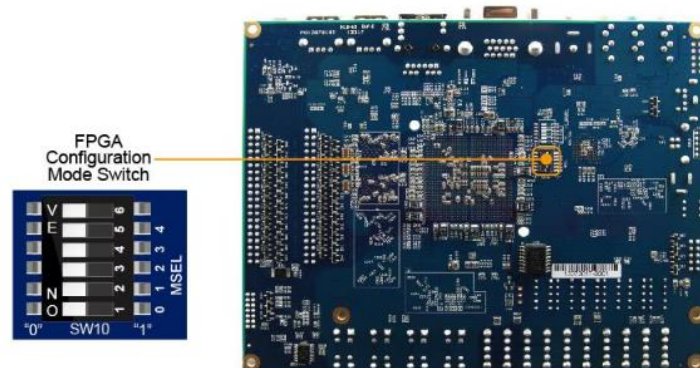


Figura 3.26 Configuración del selector MSEL de la tarjeta DE1 SoC, [22].

Con dicha configuración ya se puede iniciar el FPGA, solo se requiere conectar un monitor con entrada VGA, un mouse y un teclado con entrada USB y la microSD con el sistema operativo y el archivo soc_system.rbf guardado previamente en ella. El diagrama de conexiones de los equipos es el que se muestra en la Figura 3.27.

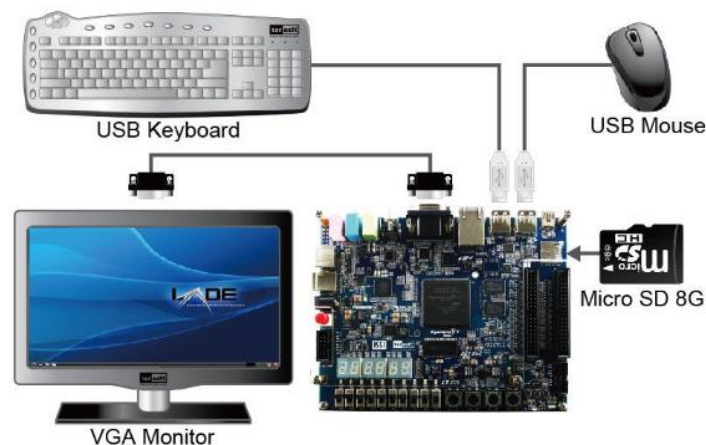


Figura 3.27 Conexiones y equipo requerido para utilizar el FPGA en modo gráfico.

Al encender el FPGA en la parte superior izquierda de la pantalla aparecerá el icono de Linux (pingüino), mostrando así que el sistema operativo arrancó y que el HPS está enlazado con el FPGA.

CAPÍTULO 4: INTERFAZ GRÁFICA DE USUARIO

4.1 INTRODUCCIÓN

En este capítulo, se muestra cómo utilizar el JNI para que una interfaz gráfica realizada en Java pueda utilizar códigos de C, los cuales por medio de punteros interactuaran con las localidades de memoria del procesador a los cuales están conectadas las entradas y salidas de una arquitectura FPGA que se quieran controlar o leer.

4.2 LENGUAJE C

El lenguaje de programación C fue creado por Brian Kernighan y Dennis Ritchie a mediados de los años 70. El lenguaje C es un lenguaje para programadores en el sentido de que proporciona una gran flexibilidad de programación y una muy baja

comprobación de incorrecciones, de forma que el lenguaje deja bajo la responsabilidad del programador acciones que otros lenguajes realizan por sí mismos.

El lenguaje C es un lenguaje estructurado, pero no es estructurado por bloques, o sea, no es posible declarar subrutinas (pequeños trozos de programa) dentro de otras subrutinas, a diferencia de como sucede con otros lenguajes estructurados. Además, el lenguaje C no es rígido en la comprobación de tipos de datos, permitiendo fácilmente la conversión entre diferentes tipos de datos y la asignación entre tipos de datos diferentes [23].

Todo programa de C consta, básicamente, de un conjunto de funciones y una función llamada main, la cual es la primera que se ejecuta al comenzar el programa, llamándose desde ella al resto de funciones que componga el programa.

Desde su creación, surgieron distintas versiones de C, que incluían unas u otras características, palabras reservadas, etc. Este hecho provocó la necesidad de unificar el lenguaje C, y es por ello que surgió un standard de C, llamado ANSI-C, que declara una serie de características, etc., que debe cumplir todo lenguaje C. Por ello, y dado que todo programa que se desarrolle siguiendo el standard ANSI de C será fácilmente portable de un modelo de ordenador a otro modelo de ordenador, y de igual forma de un modelo de compilador a otro.

4.2.1 PUNTEROS EN C

La forma en la que se pueden controlar y leer señales de una arquitectura de un FPGA SoC, es por medio de las localidades de memoria del procesador. El procesador tiene una sección de su memoria designada para poder interconectar el FPGA con él, dicha sección es llamada "Lightweight H2F Bridge". En la hoja de datos se puede encontrar la base address del h2f_lw bridge (0xFF200000). También se encuentra la address región, la cual tiene un tamaño de 2MB (0x00200000). Cuando se realizó el enlace entre FPGA y HPS todas las señales del FPGA conectadas al HPS se conectaron a alguna dirección de memoria de dicha address region [24].

Para poder acceder a dichas localidades de memoria y poder leer lo que contienen o poder escribir sobre ellas, se utilizan punteros. En el famoso libro de Kernighan y Ritchie “El lenguaje de programación C”, se define un puntero como «una variable que contiene la dirección de una variable». Tras esta definición clara y precisa, se puede remarcar que un puntero es un tipo especial de variable que almacena el valor de una dirección de memoria [25].

El lenguaje de programación C es uno de los que te permiten manejar de una manera muy sencilla los punteros, así que con un programa en C se pueden leer y controlar las señales de un FPGA conectado a la memoria de un HPS en un sistema SoC.

4.3 JAVA

Java es un lenguaje de programación creado por Sun Microsystems, (una compañía que luego fue comprada por Oracle) para poder trabajar en diferentes tipos de procesadores. Su sintaxis es muy similar a la de C o C ++ e incorpora como propias algunas características que en otros lenguajes son extensiones. El código Java, una vez compilado, puede transportarse sin modificación en cualquier máquina y ejecutarlo. Esto se debe a que el código se ejecuta en una máquina virtual o hipotética, la Máquina virtual Java, que es responsable de interpretar el código (archivos .class compilados) y convertirlo a un código particular de la CPU que se está utilizando (siempre que dicha máquina virtual sea compatible) [26].

Java es un lenguaje orientado a objetos (OO). El elemento fundamental cuando se habla de programación orientada a objetos es el concepto de objeto en sí mismo, así como el concepto abstracto de clase. Un objeto es un conjunto de variables junto con los métodos relacionados con ellas. Contiene la información (las variables) y la forma de manipular la información (los métodos). Una clase es el prototipo que define las variables y los métodos que utilizará un determinado tipo de objeto, es la definición abstracta de lo que asume un objeto en la memoria.

4.3.1 JAVA AWT Y SWING

AWT (Abstract Window Toolkit) es una librería de clases Java para el desarrollo de interfaces de usuario gráficas (GUI). Por tratarse de código Java, las aplicaciones serán independientes de la plataforma. No así su apariencia visual, que será el de la plataforma donde se ejecute. AWT es la librería visual básica. Sobre ella se construyó a posteriormente otra más flexible y potente: Swing, [27].

Swing se describe como un conjunto de componentes gráficos personalizables cuya apariencia se puede dictar en tiempo de ejecución. Swing es el kit de herramientas GUI de próxima generación que Sun Microsystems creó para permitir el desarrollo empresarial en Java. Por desarrollo empresarial, se refiere a que los programadores pueden usar Swing para crear aplicaciones Java a gran escala con una amplia gama de componentes potentes. Además, puede ampliar o modificar fácilmente estos componentes para controlar su apariencia y comportamiento [28].

4.3.2 JNI

La JNI (Java Native Interface) es una característica poderosa de la plataforma Java. Las aplicaciones que usan el JNI pueden incorporar códigos de escritura escrita en lenguajes de programación como C y C++, así como código escrito en el lenguaje de programación de Java. El JNI permite a los programadores aprovechar el poder de la plataforma Java, sin tener que abandonar sus inversiones en código heredado. Como el JNI es parte de la plataforma Java, los programadores pueden abordar los problemas de interoperabilidad una vez y esperar que su solución funcione con todas las implementaciones de la plataforma Java [29].

Cuando la plataforma Java se implementa sobre algunos entornos, puede ser deseable o necesario permitir que las aplicaciones Java trabajen estrechamente con códigos nativos escritos en otros idiomas. Los programadores han comenzado a adoptar la plataforma Java para construir aplicaciones que tradicionalmente se escribieron en C y C++. Sin embargo, debido a la inversión existente en código heredado, las aplicaciones Java coexistirán con el código C y C++ durante muchos años. El JNI es una

característica poderosa que le permite aprovechar la plataforma Java, pero aún utilizar código escrito en otros idiomas. Como parte de la implementación de la máquina virtual Java, el JNI es una interfaz bidireccional que permite que las aplicaciones Java invoquen código nativo y viceversa. En la Figura 4.1 se puede observar cómo funciona JNI enlazando códigos nativos (Lenguaje C) con una aplicación hecha en Java.

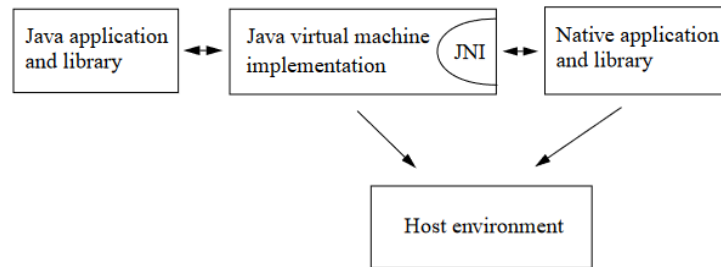


Figura 4.1 Rol del JNI en una aplicación, [29].

El JNI está diseñado para manejar situaciones en las que necesita combinar aplicaciones Java con código nativo. Como interfaz bidireccional, el JNI puede admitir dos tipos de código nativo: bibliotecas nativas y aplicaciones nativas.

Cuando una aplicación usa el JNI, se corre el riesgo de perder dos beneficios de la plataforma Java:

1. Primero, las aplicaciones Java que dependen del JNI ya no pueden ejecutarse fácilmente en múltiples entornos host. A pesar de que la parte de una aplicación escrita en el lenguaje de programación Java es portátil a múltiples entornos host, será necesario volver a compilar la parte de la aplicación escrita en lenguaje de programación nativo.
2. Segundo, mientras que el lenguaje de programación Java es de tipo seguro y seguro, lenguajes nativos como C o C ++ no lo son. Como resultado, debe tener especial cuidado al escribir aplicaciones utilizando el JNI. Un método nativo que se porta mal puede dañar toda la aplicación. Por esta razón, las aplicaciones Java están sujetas a controles de seguridad antes de invocar características JNI.

En este caso el JNI es requerido, ya que para tener acceso a la conexión entre el HPS y el FPGA, se tiene que hacer por medio de un programa en C que maneje punteros,

Java fungirá como interfaz gráfica de usuario que mande a llamar códigos en C por medio de JNI que interactúen con el FPGA.

4.3.3 ECLIPSE

Eclipse es un entorno de desarrollo integrado (IDE) utilizado en la programación informática y es el IDE de Java más utilizado. Contiene un espacio de trabajo base y un sistema de complemento extensible para personalizar el entorno. Eclipse está escrito principalmente en Java y su uso principal es para desarrollar aplicaciones Java, pero también puede usarse para desarrollar aplicaciones en otros idiomas de programación mediante el uso de complementos [30].

Eclipse proporciona IDE y plataformas para casi todos los lenguajes y arquitecturas. Son famosos por sus IDE de Java, C/C++, JavaScript y PHP creados en plataformas extensibles para escritorio, web o en la nube. Estas plataformas ofrecen la colección más extensa de herramientas adicionales disponibles para desarrolladores de software.

Eclipse favorece a los programadores porque ayuda en el proceso de desarrollo al proporcionar las siguientes características clave:

- Una interfaz gráfica de usuario fácil de usar que navega a través de su jerarquía de código.
- El resaltado de sintaxis que muestra el código fuente en un formato de código de color para mejorar la legibilidad. Finalización de código que hace recomendaciones sobre métodos y parámetros a medida que escribe.
- Recomendaciones sobre cómo corregir errores y corrección automática de errores.
- Un depurador gráfico que permite la inspección de código línea por línea.
- Compilación y ejecución de un solo programa de clave.
- Generación automática de código para patrones de uso común.
- Integración con repositorios de control de versión de código fuente.

Para ver como instalar eclipse revisar Apéndice D [31].

4.4 CONTROLAR Y LEER SEÑALES DE UN SoC FPGA MEDIANTE UNA INTERFAZ GRÁFICA DE USUARIO

El desarrollo de la interfaz se realiza en primer término en una PC con el fin de ahí programar lo que sea necesario y después solo pasar todo el código al dispositivo final que será la DE1 SoC con Linux integrado, donde se correrá lo realizado en el primer dispositivo.

Para poder usar las funciones que ofrece JNI en eclipse primero se tiene que instalar CDT (C/C++ Development Tool) en eclipse (Apéndice F).

Una vez instalado el CDT se creó el código que se utilizó para hacer la interfaz gráfica. El código se divide en 5 partes las cuales son: la interfaz gráfica, la acción de un cuadro combinado para seleccionar que espectro de señales mostrar, la acción de un cuadro combinado para seleccionar que componentes armónicas mostrar, la acción de un cuadro combinado para seleccionar el nivel de tensión al que se realice la medición y la acción de un botón el cual se encargará de tomar las muestras obtenidas por el ADC, desplegarlas en una gráfica, así como un espectro de frecuencias, además de con las muestras obtenidas calcular todos los parámetros eléctricos.

La parte de la interfaz gráfica funciona en base a los layouts que ofrece la programación en Java, se ponen un layout principal y en los demás por medio de paneles que llevan sus propios layouts. El layout principal es un BorderLayout donde se utilizarán la parte “NORTH” y “CENTER”. En la Figura 4.2 se muestra como están divididas las partes de un BorderLayout.

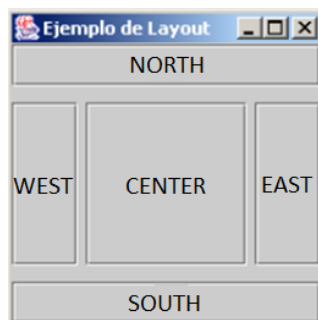


Figura 4.2 BorderLayout de Java Swing.

La parte “NORTH” contendrá un FlowLayout en donde se colocarán: un cuadro de texto para seleccionar la frecuencia de muestreo, un cuadro de texto para seleccionar el número de muestras a adquirir, un cuadro combinado para seleccionar de que señal mostrar el espectro de frecuencias, un cuadro combinado para seleccionar que componentes armónicos mostrar, un cuadro combinado para seleccionar a que nivel de tensión se realizará la medición y un botón para adquirir las muestras y el espectro de frecuencias, desplegarlos y realizar los cálculos de los parámetros eléctricos. En la Figura 4.3 se muestra un ejemplo de un FlowLayout, mientras que en la Figura 4.4 se muestra como se aplicó para la interfaz gráfica que se desarrollo.

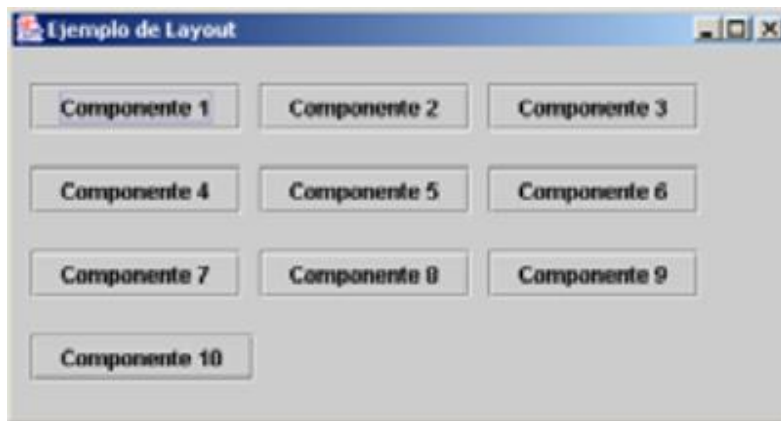


Figura 4.3 FlowLayout de Java Swing.

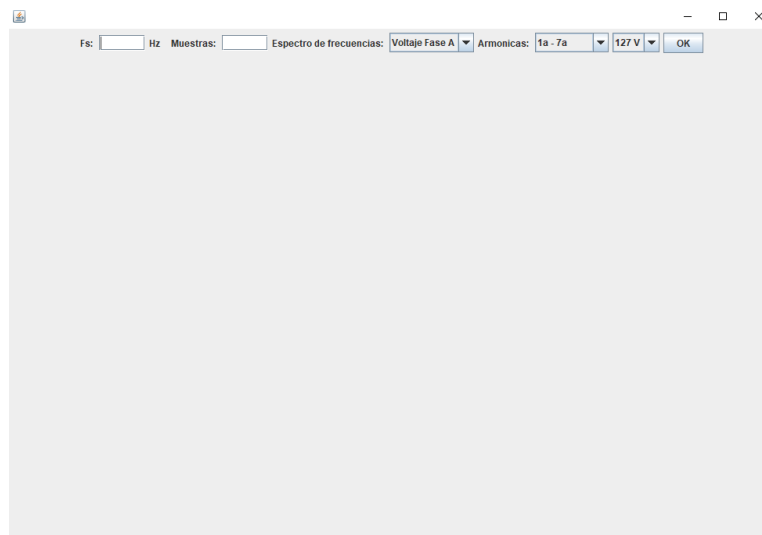


Figura 4.4 FlowLayout aplicado en la parte “NORTH” del BorderLayout.

En la parte de “CENTER” del BorderLayout se aplicó un GridLayout de 2x2, en la Figura 4.5 se puede observar cómo funciona un GridLayout de 2x3. En las primeras 3 posiciones del GridLayout a utilizar estarán 3 gráficas que son parte de una librería extra llamada JFreeChart, la cual se puede descargar de su página oficial [32]. Para ver como agregar librerías a eclipse (ver Apéndice G). En la Figura 4.6 se puede observar cómo quedaría la interfaz gráfica con las tres gráficas añadidas.

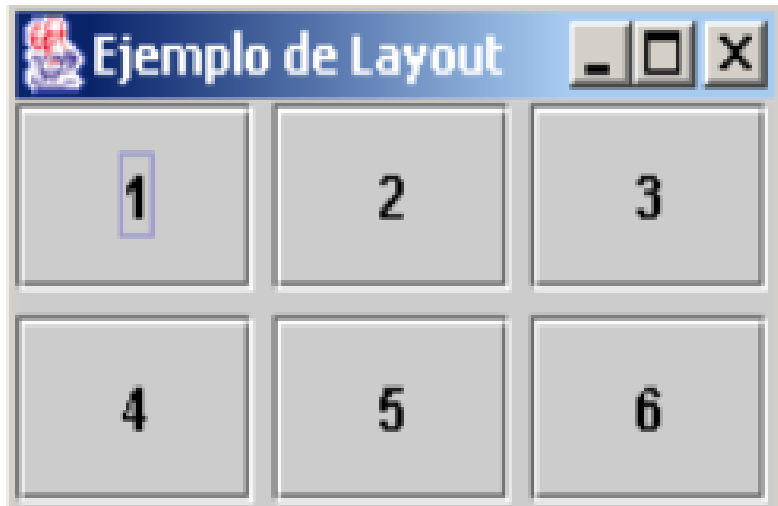


Figura 4.5 Ejemplo de GridLayout de 2x3.

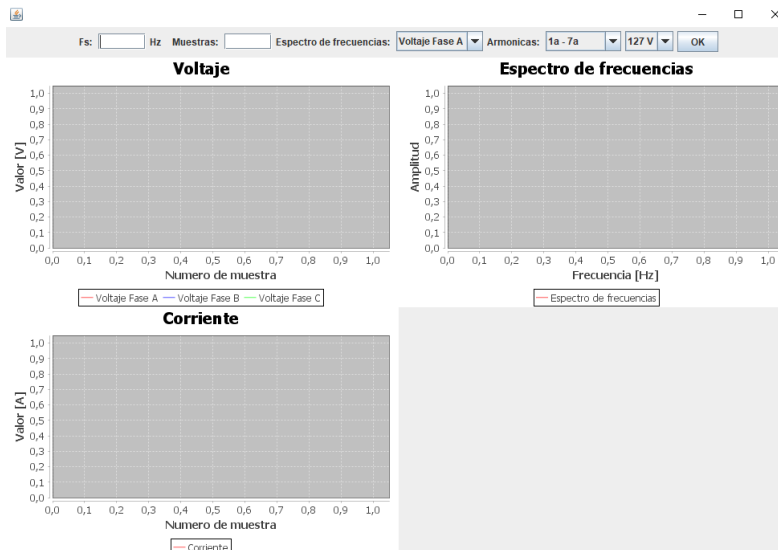


Figura 4.6 GridLayout aplicado a la parte “CENTER” de un BorderLayout.

Tres de los 4 cuadros generados por el GridLayout llevarán gráficas y el último cuadro contendrá en el otro GridLayout de 1x2, que a su vez cada separación llevará en él un GridLayout de 8x1 donde en el primer GridLayout se colocarán los parámetros eléctricos calculados y en el segundo el porcentaje con el que contribuyen los armónicos de la señal que esté mostrando su espectro de frecuencias. La interfaz gráfica con todos sus componentes puede ser observada en la Figura 4.7.

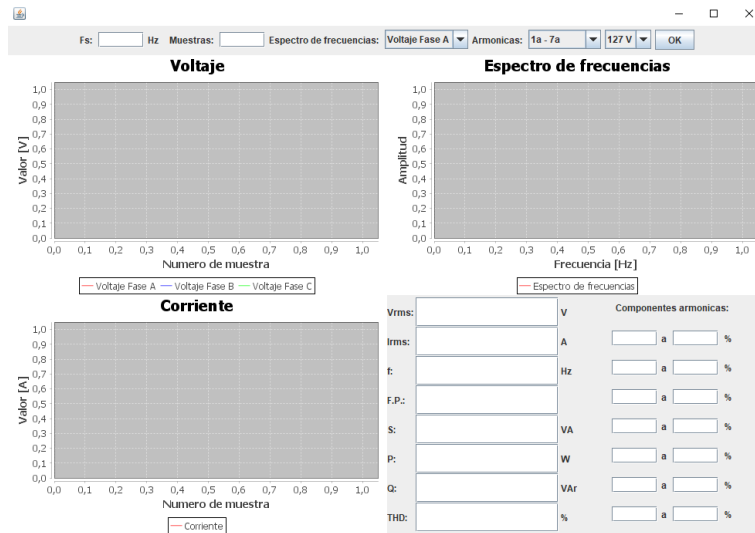


Figura 4.7 Interfaz gráfica programada.

El núcleo del código y que hace que este funcione es la acción del botón que gráfica, obtiene los datos y calcula los parámetros eléctricos. Lo primero que hace esa parte del código es adquirir lo escrito en los cuadros de texto (Frecuencia de muestreo y Número de muestras), con la frecuencia de muestreo se obtiene el número que será enviado a la entrada de control del FPGA, el cual se obtiene con la ecuación 9 (Página 26).

El número de muestras sirve para delimitar los ciclos for que obtendrán las muestras y además de dar el tamaño de unos arreglos que funcionan para almacenar en ellos los valores adquiridos por el ADC, también hay arreglos que funcionan para almacenar en ellos las FFT adquiridas pero estos ya tienen un tamaño fijo de 256 ya que la FFT que se está utilizando es una de 512 muestras y solamente la primera mitad de los datos es rescatable para mostrarla en el espectro de frecuencias.

Se cuenta con condicionales de seguridad para que no acepte valores mayores a 123762 Hz de frecuencia de muestreo, ya que está es la frecuencia máxima de muestreo obtenible por el ADC, además, que no pase de 4200 muestras ya que las especificaciones del sistema indican que el equipo debe ser capaz de mostrar 2 ciclos de 60Hz a la frecuencia de muestreo máxima, si se sabe que el número de muestras de un ciclo está en función de su frecuencia y su frecuencia de muestreo, y se obtiene con la siguiente ecuación:

$$n = \frac{f_s}{f} \quad (107)$$

donde: n es el número de muestras en un ciclo de la señal, f_s es la frecuencia de muestreo a la cual se adquiere la señal y f es la frecuencia fundamental de la señal. Con la ecuación 107 se obtiene que una señal de 60Hz muestreada a 123762 Hz se puede adquirir en:

$$n = \frac{123762}{60} = 2062 \text{ muestras} \quad (108)$$

Por lo que 2 ciclos de 60Hz muestreados a 123,762Hz se obtendrán en 4125 muestras. Se dejó 4200 para mostrar un poco más de dos ciclos. También se puso como límite inferior el 0 para ambos valores (Frecuencia de muestreo y número de muestras).

Una vez que los parámetros de entrada están en los rangos permitidos se habilita la adquisición del FPGA y se adquieren el número de muestras seleccionadas, en esta parte también se almacena en el arreglo correspondiente cada que se adquiere una muestra, se gráfica y antes de graficar se escala el valor adquirido. El valor adquirido es escalado a 127 V o 220 V según lo que se tenga seleccionado en la caja combinada, por default está a 127 V .

Cuando se terminan de adquirir las muestras de las 4 señales, se grafican y se almacenan en sus respectivos vectores, se entra a un ciclo del cual no se puede salir hasta que el FPGA mande la señal de que la FFT fue calculada y almacenada en la memoria FIFO.

Cuando el FPGA haya terminado de calcular la FFT se obtendrán los valores del espectro de frecuencias de cada señal y también se almacenarán cada uno en su respectivo arreglo, se ajustarán los datos y se graficarán según el espectro seleccionado en la caja combinada, por default se grafica el espectro de frecuencias del Voltaje Fase A.

Una vez almacenadas y graficadas los resultados de la adquisición y de sus FFT se deshabilita el sistema de adquisición y se pasa al cálculo de los parámetros eléctricos.

Todos los cálculos se realizan con los arreglos en los que se almacenaron los datos que mandó el FPGA. El primer parámetro que se calcula es la frecuencia, pero antes de eso, se tiene una detección de DC, si la lectura del ADC es constante, se detecta con el fin de que los cálculos de la frecuencia no den un frecuencia igual a 0 y al momento de hacer cálculos se divida entre 0. En caso de que haya corriente directa no se realizará ningún cálculo. La detección es independiente de cada señal por lo que en caso de que no se conecte alguna señal se pueda hacer cálculos con las demás.

Una vez que se detecta que la señal no es constante se procede al cálculo de la frecuencia, para esto el método que se utiliza es el siguiente: primero se encuentran el valor más alto y más bajo que haya sido registrado en cada señal con dichos valores se puede obtener un punto medio el cual será el “cero” de la señal. Después se hace un barrido sobre las muestras y se encuentra el primer cruce por ese cero, el cruce que se detecta es el que va de menor a mayor, es decir cuando hay un dato menor a ese cero y el siguiente es mayor, no se detectan los cruces que van de mayor a menor.

En la Figura 4.8 se puede observar cómo funciona el algoritmo de detección de cruces por cero, marcando con un check los cruces que son detectables y con una cruz los no detectados, también con una línea naranja el mayor dato detectado, con una línea azul el menor dato detectado y con una línea amarilla el “cero” detectado.



Figura 4.8 Cruces por ceros que pueden ser detectados con el algoritmo.

El siguiente paso en el cálculo de la frecuencia, es monitorear la frecuencia con ayuda del espectro de frecuencias. En un espectro de frecuencias obtenido por medio de una FFT el eje de las ordenadas se representa la magnitud con la que una frecuencia contribuye a la señal y en las abscisas están las frecuencias encontradas con la FFT, el espectro de frecuencias es una gráfica discreta por lo que cuenta con un rango entre dato obtenido y dato obtenido, dicho rango depende de la frecuencia de muestreo y del tamaño de la FFT y se obtiene con la siguiente formula:

$$\text{rangoFFT} = \frac{f_s}{\text{TamañoFFT}} \quad (109)$$

En este caso se tiene un tamaño de 512 por lo que la ecuación 109 se puede escribir de la siguiente forma:

$$\text{rangoFFT} = \frac{f_s}{512} \quad (110)$$

Para saber qué frecuencia representa cada muestra obtenida de la FFT, solamente hay que multiplicar dicho rango por el número de muestra.

$$f = n \times \text{rangoFFT} = n \frac{f_s}{512} \quad (111)$$

Lo que realiza a continuación es encontrar el número de muestra de la FFT que tenga el valor más alto, ya que en esa frecuencia se encontrará la frecuencia fundamental del sistema según la FFT. Esta frecuencia calculada con ayuda de la FFT solo servirá como guía para aproximarse a la frecuencia verdadera del sistema, ya que sabiendo la

frecuencia que podría ser la fundamental, se puede saber en cuantas muestras podría estar la verdadera frecuencia.

Hasta este momento se tiene detectado el primer cruce por cero y el número de muestras en los que se podría repetir la señal. Si se suman dichos datos, en teoría se debería de poder aproximar al siguiente cruce por cero y así se realiza, comenzando desde ahí a buscar el siguiente cruce por “cero”.

Cuando se tienen los dos cruces por “cero”, solo bastará restar el número de muestra en el que se encuentra cada uno para saber en cuantas muestras paso un ciclo y con ayuda de la formula siguiente encontrar el valor de la frecuencia de la señal.

$$f = \frac{f_s}{n} \quad (112)$$

donde:

$$n = \text{segundo cruce por cero} - \text{primer cruce por cero} \quad (113)$$

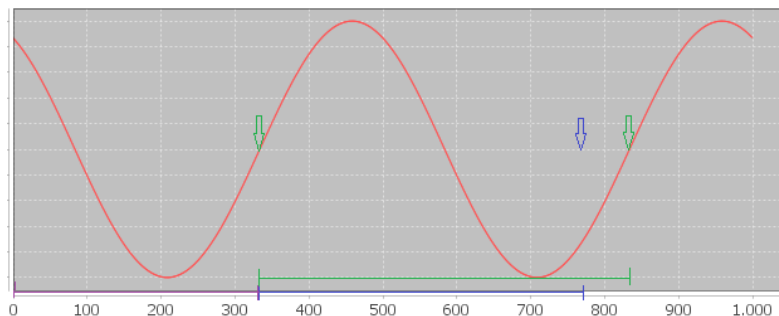


Figura 4.9 Cruces por cero detectados y cruce por cero aproximado por la FFT.

En la Figura 4.9 se puede observar en verde los cruces por cero detectados por el algoritmo y en azul un aproximado calculado con ayuda del espectro de frecuencias.

Con el cálculo de la frecuencia es posible calcular los parámetros eléctricos. Los cálculos se hacen sobre el Voltaje Fase A y su corriente, ya que es la única que se adquiere con el ADC. Primero se calculan los valores eficaces, como ya se conoce el número de muestras en los que pasa un ciclo es posible conocer el valor eficaz, en este caso se al ser una señal discreta se calcula con la siguiente ecuación:

$$V_{RMS} = \frac{1}{n} \sqrt{\sum_{k=0}^{n-1} V[k]^2} \quad (114)$$

donde: n es el número de muestras en las que pasa un ciclo, y V[k] es el arreglo donde se almacenaron las n muestras adquiridas por el ADC.

Para obtener la potencia aparente, primero se tiene que comprobar que la señal de voltaje y de corriente tengan la misma frecuencia fundamental, si es así se podrá calcular simplemente multiplicando el voltaje eficaz y la corriente eficaz.

$$S = V_{RMS} I_{RMS} \quad (115)$$

El parámetro siguiente a calcular es el factor de potencia el cual se obtiene con ayuda de los cruces por cero detectados anteriormente para el Voltaje Fase A y la corriente, lo que se hace es restar ambos valores, la resta entre dichas muestras es el desfaseamiento, pero al conocer la frecuencia de muestreo es posible conocer ese desfaseamiento en tiempo, lo cual se puede traducir en radianes según la frecuencia obtenida anteriormente. El desfaseamiento en tiempo se calcula con la siguiente ecuación:

$$\phi = \frac{n}{f_s} \quad (116)$$

donde: ϕ es el desfaseamiento, n es el número de muestras entre un cruce por cero y otro y f_s la frecuencia de muestreo.

Para convertir de segundos a radianes se utiliza la siguiente ecuación:

$$\phi[\text{rad}] = \phi[\text{s}] \left(\frac{2\pi \text{ rad}}{T} \right) = \frac{(2\pi)(f)(n)}{f_s} \quad (117)$$

donde: f es la frecuencia fundamental del sistema calculada anteriormente.

Por definición para obtener el factor de potencia solamente hay que sacar el coseno del ángulo obtenido, sabiendo que el ángulo debe cumplir la siguiente condición:

$$-90^{\circ} \leq \phi \leq 90^{\circ} \quad (118)$$

En caso de no cumplirse, se deberá revisar la polaridad de los instrumentos de medición ya que estos podrían desfasar las señales 180° y haga que no sea posible calcular el factor de potencia.

$$FP = \cos(\phi) \quad (119)$$

Con el ángulo de desfase y el factor de potencia es posible calcular las potencias real y reactiva con ayuda de las siguientes formulas:

$$P = S(FP) \quad (120)$$

$$Q = S \sin(\phi) \quad (121)$$

Para los componentes armónicos, lo que se realizó fue utilizar el valor más alto del espectro de frecuencias obtenido con anterioridad y utilizarlo como componente principal. Y con ayuda del espectro de frecuencias encontrar las componentes armónicas de dicho elemento y almacenarlas en arreglos.

La THD se calcula en base a los valores almacenados en los arreglos de componentes armónicos con la siguiente fórmula:

$$THD = \frac{\sqrt{\sum_{k=1}^m A[k]^2}}{A[0]} \quad (122)$$

donde: m es el número de componentes armónicas detectadas y A[] es el arreglo donde se almacenaron las componentes armónicas.

Para su despliegue depende de la selección de las cajas combinadas (Selección de espectro de frecuencias y de componentes armónicos).

El funcionamiento de ambas cajas es similar, lo que hacen es guardar en una variable la selección y desplegar en base a los arreglos donde están almacenados tanto el espectro de frecuencias como los componentes armónicos que estén seleccionados. La frecuencia y la THD también cambian con respecto a dicha selección.

Para poder hacer uso del JNI se debe crear un a clase que invoque el código en C que funcionará como una librería. [33]

El código que invoca al JNI lo único que hace es crear unas funciones utilizables en Java que sacarán su contenido de una librería nativa en lenguaje C llama JNI_Lib.

Para poder crear la librería que se leerá con el código anterior se debe convertir el proyecto a uno de C/C++, para esto se deberá oprimir el botón guardar en ambas clases (interfaz gráfica y generadora de funciones). Luego sobre la carpeta del proyecto se debe oprimir el botón derecho del ratón y luego oprimir el botón New >> Other tal como se muestra en la Figura 4.10.

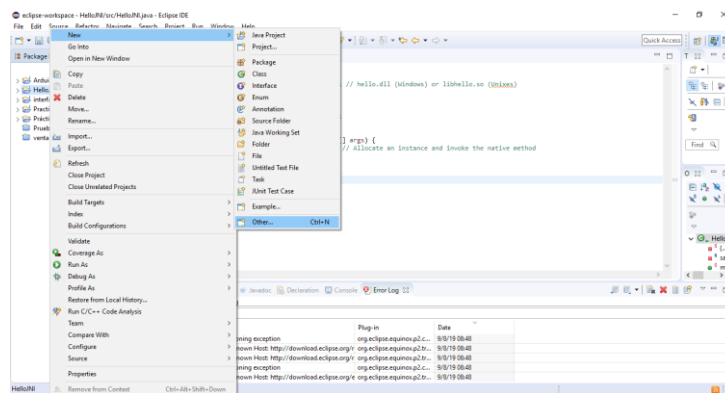


Figura 4.10 Primer paso para convertir un proyecto a un proyecto de C/C++.

Esto abrirá una ventana como la vista en la Figura 4.11, en la cual se debe expandir la carpeta llamada “C/C++” y seleccionar la opción “Convert to a C/C++ Project (Adds C/C++ Nature)” y luego oprimir el botón Next.

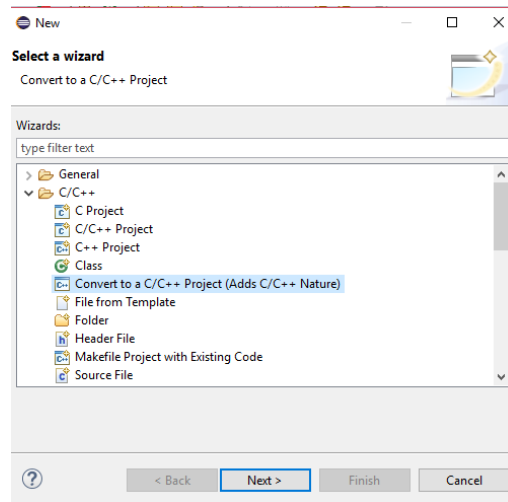


Figura 4.11 Convertir un proyecto a proyecto de C/C++.

Ahora se debe seleccionar el proyecto, seleccionar la opción “C Program”, en Project type seleccionar “Makefile project” y en Toolchains seleccionar “Cygwin GCC”, Finalmente oprimir el botón finish. Las configuraciones anteriores se muestran en la Figura 4.12.

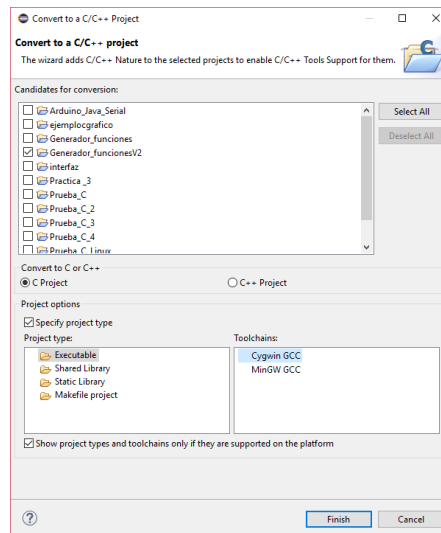


Figura 4.12 Configuración del proyecto de C.

Dentro de la carpeta del proyecto se debe crear una carpeta donde se almacenarán los códigos de C, para crearla se oprime el botón derecho del ratón en la carpeta del proyecto luego se selecciona New >> Folder. El nuevo folder llevará el nombre de “jni” y puede observarse en la Figura 4.13.

Se abrirá una ventana en la que se nombrará a un nuevo target, este llevara el nombre de “JNIserver.h”, en la Figura 4.15 se muestra como configurar el nombre del target.

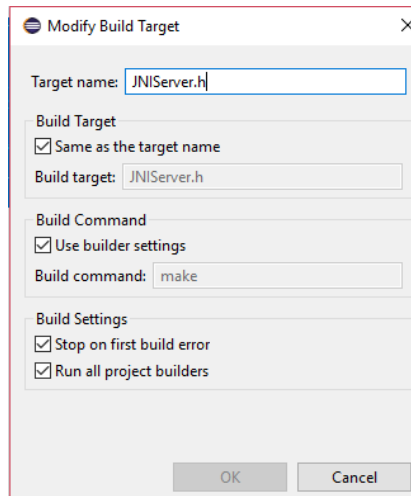


Figura 4.15 Configuración del nuevo target.

Para correr el makefile primero se debe abrir la carpeta del proyecto llamada bin y seleccionar el archivo llamado “JNIserver.class”, luego oprimir el botón la pestaña Run >> External Tools >> External Tools Configuration tal y como se muestra en la Figura 4.16.

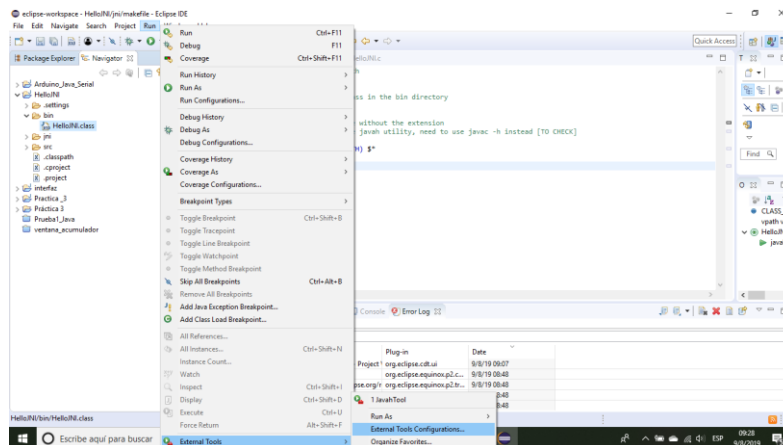


Figura 4.16 Habilitación del comando Javah.

Esto abrirá una ventana como la que se observa en la Figura 4.17, donde en el espacio Name se pondrá “JavahTool”, en la parte de location se deberá poner la dirección donde este guardado el archivo llamado “javah.exe” el cual se encuentra en la carpeta

de instalación del JDK, en este caso se encuentra en “C:\Program Files (x86)\Java\jdk1.8.0_202\bin\javah.exe”, en la parte de Working Directory se oprimirá el botón el botón que dice “Browse Workspace” y ahí se abrirá una ventana como la que se muestra en la Figura 4.18 y se seleccionará la carpeta llamada “bin” que se encuentra en la carpeta del proyecto, finalmente en la parte de Arguments se debe escribir el siguiente código:

```
-jni -verbose -d "${project_loc}${system_property:file.separator}jni"  
${java_type_name}
```

Se oprimirá el botón Apply y luego en Run [34]. La configuración completa se puede observar en la Figura 4.19.

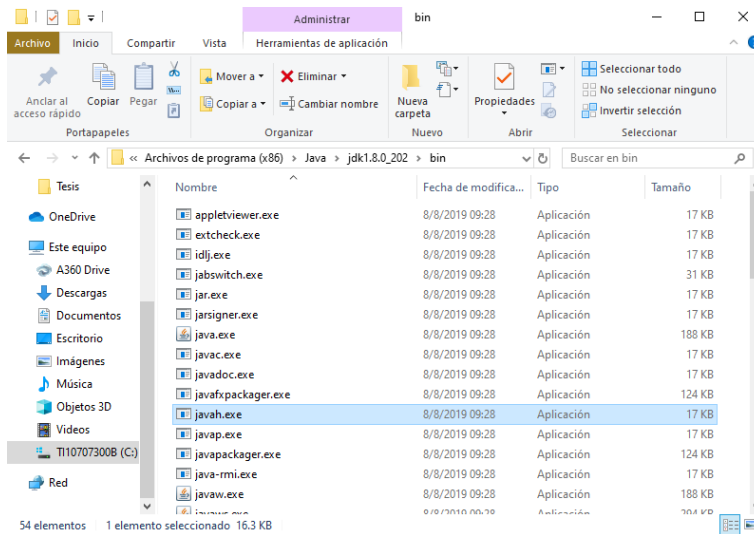


Figura 4.17 Ubicación de javah.exe.

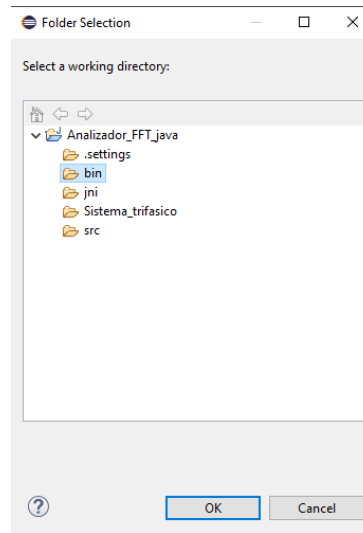


Figura 4.18 Selección del Workspace Directory.

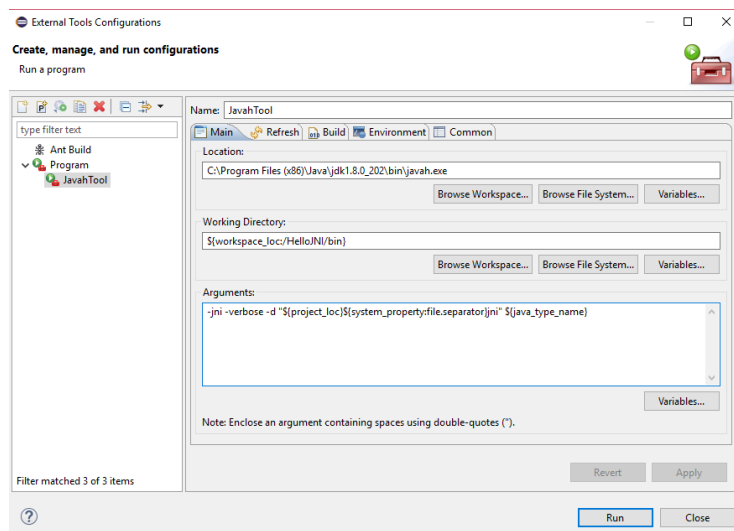


Figura 4.19 Configuración de la herramienta externa Javah.

El paso anterior solo habilitará el comando “javah” escrito en el makefile, para generar el archivo “JNIserver.h” se tiene que oprimir el botón derecho del ratón sobre el Target y seleccionando la opción “Build Target” como se muestra en la Figura 4.20.

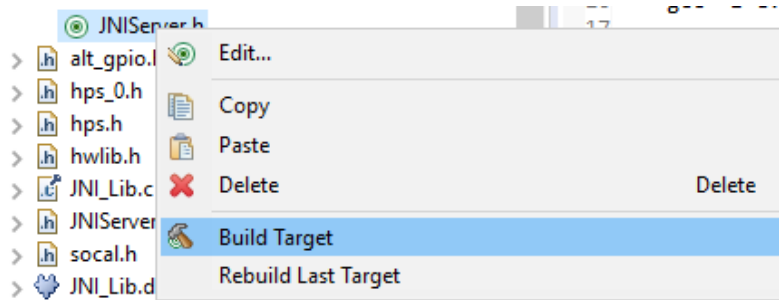


Figura 4.20 Instrucción de construir un target.

Ahora se creará la librería en lenguaje C que se invocará por medio del programa en Java, para hacer esto se debe oprimir el botón derecho del ratón sobre la carpeta llamada “jni” y oprimir el botón New >> Other y se abrirá una ventana en la cual en la carpeta llamada “C/C++” se deberá seleccionar la opción llamada “Source File” y oprimir el botón Next como se observa en la Figura 4.21.

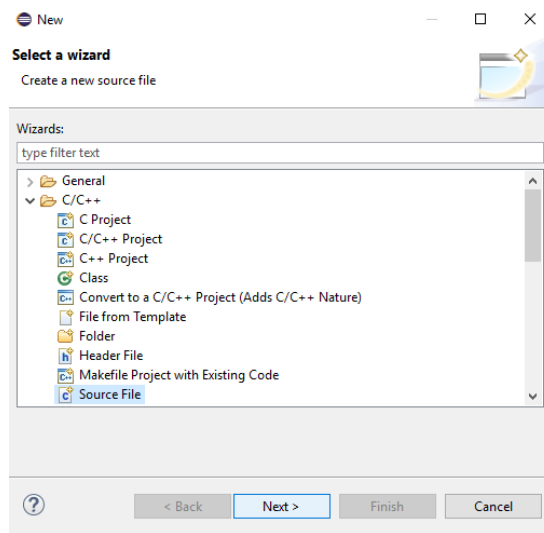


Figura 4.21 Creación de un programa en C.

A este archivo se le nombrará “JNI_Lib.c” y se oprimirá el botón Finish como se muestra en la Figura 4.22.

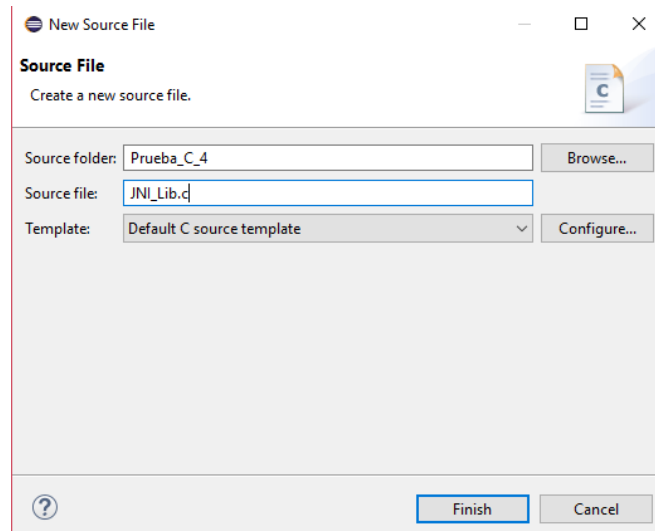


Figura 4.22 Configuración inicial del programa en C.

Ahora se deberá escribir el código en C que por medio de punteros controle o lea señales del FPGA. Para que el código funcione hay que incluir en él incluir las librerías a la carpeta jni del proyecto ya que hasta este punto solo se tiene “JNI Server.h” generada con anterioridad usando el javah. Las librerías que se deben copiar y pegar (no mover) en la carpeta JNI son: “hwlib.h”, “social.h”, “hps.h”, “alt_gpio.h” y “hps_0.h”.

Para obtener esas librerías solo se tiene que usar el SoC Embedded Command Shell. Para ver como instalarlo revisar Apéndice H.

En la carpeta de instalación del Command Shell se encuentran las primeras cuatro librerías, en este ejemplo la dirección donde se encuentra la “hwlib.h” es:

```
“C:\altera\15.1\embedded\ip\altera\hps\altera_hps\hwlib\include”
```

En la Figura 4.23 se observa la carpeta anterior donde se puede encontrar la librería “hwlib.h”.

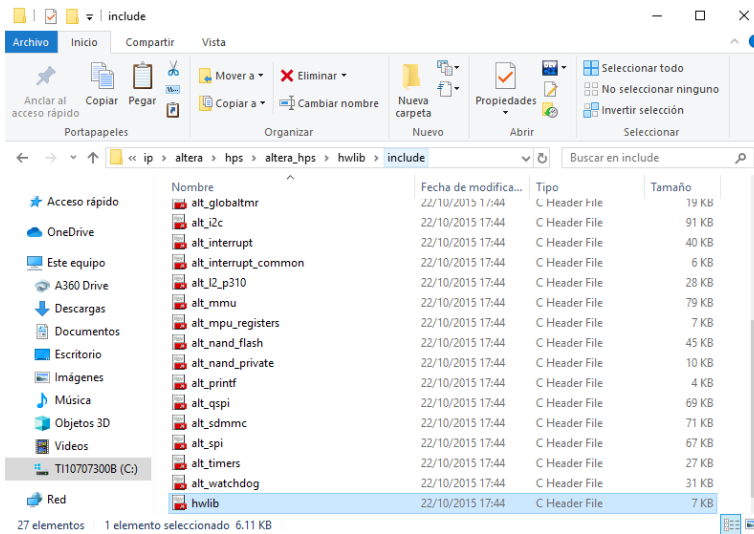


Figura 4.23 Ubicación de hwlib.h en la carpeta de instalación del SoC EDS.

Las siguientes tres librerías se encuentran en la siguiente dirección:

“C:\altera\15.1\embedded\ip\altera\hps\altera_hps\hwlib\include\soc_cv_av\socal”

En las Figuras 4.24 y 4.25 se puede observar los archivos requeridos ubicados en la misma carpeta.

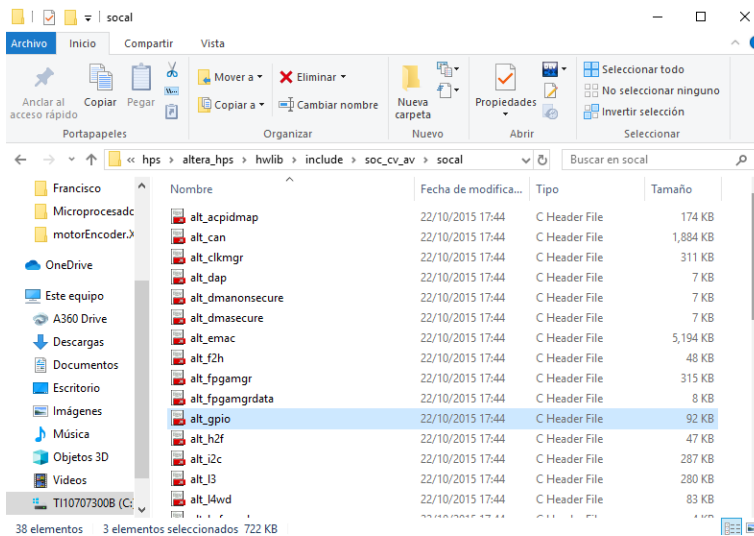


Figura 4.24 Ubicación de alt_gpio.h en la carpeta de instalación del SoC EDS.

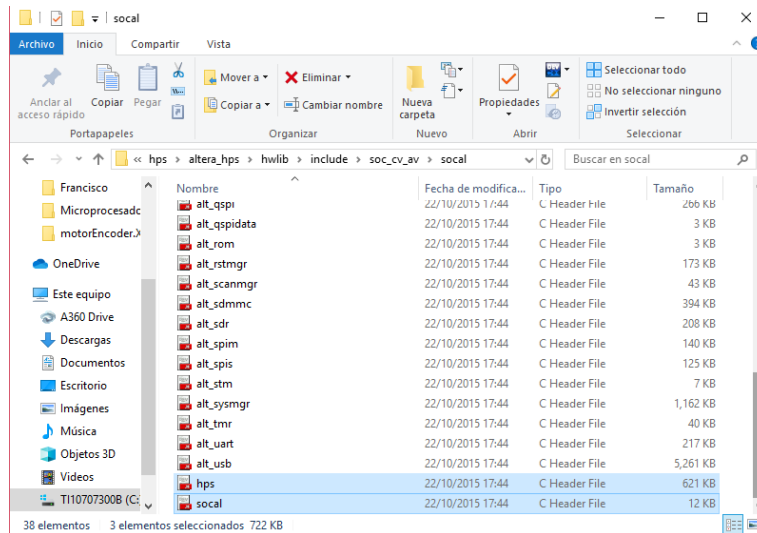


Figura 4.25 Ubicación de hps.h y social.h en la carpeta de instalación del SoC EDS.

Para la librería “hps_0.h” se deben hacer otros pasos, ya que esta es una librería generada en base al sistema que se realizó previamente en Qsys. Para obtenerla se debe abrir la consola SoC Embedded Command Shell, la cual se encuentra en un archivo por lotes en la carpeta de instalación del SoC EDS que es la mostrada en la Figura 4.26. Al abrir el archivo se abrirá una ventana como la mostrada en la Figura 4.27.

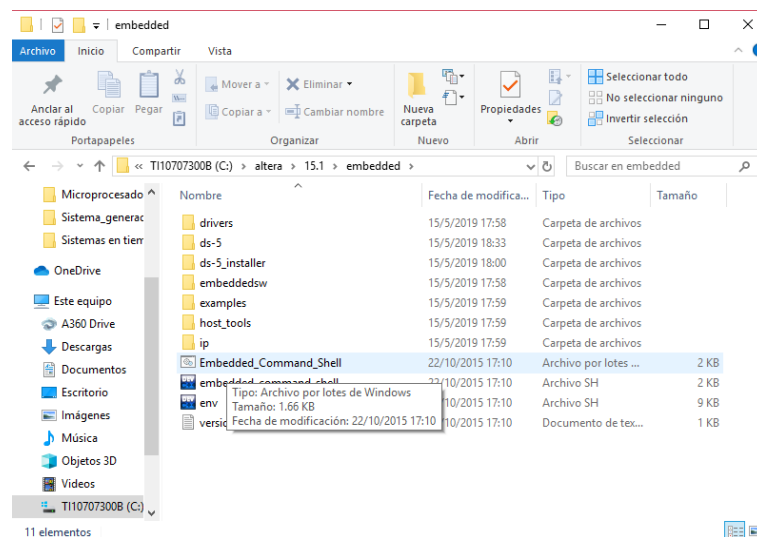


Figura 4.26 Ubicación del SoC Embedded Command Shell.

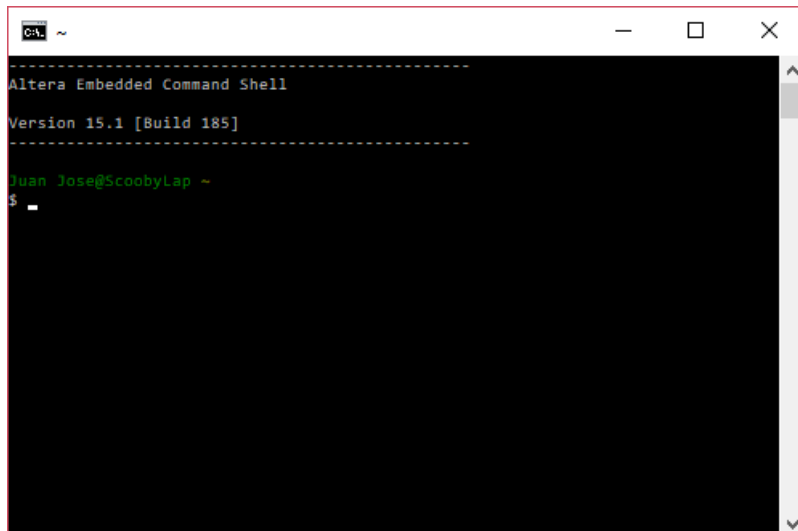


Figura 4.27 Embedded Command Shell.

Ahora se debe navegar hasta la carpeta del proyecto de Quartus (donde se encuentre el archivo de Qsys) y se debe crear un archivo "generate.sh".

Se debe navegar desde el Command Shell hasta la carpeta del proyecto de Quartus y ejecutar el archivo llamado "generate.sh". Nota: en caso de que el archivo "generate.sh" no pueda ser utilizado por un error que diga que en la línea "sopc-create-header-files no encontrado" se deberá direccionar un path a una carpeta que se encuentra en la carpeta de instalación del Quartus, esto se hace con el comando:

PATH=/cygdrive/C/altera_lite/15.1/quartus/sopc_builder/bin:\$PATH (En este caso fue así porque se tenía la versión 15.1 de Quartus en caso de tener otra versión buscar la carpeta solicitada dentro de la carpeta del Quartus instalado y poner dicha dirección) [35].

Los pasos anteriores se resumen en la Figura 4.28, donde se puede observar todos los comandos utilizados.

```

C:\cygdrive/c/Reporte
Altera Embedded Command Shell
Version 15.1 [Build 185]

jose@scobyLap ~
$ cd C:\Reporte

jose@scobyLap /cygdrive/c/Reporte
$ ls
acarreo.vhd                incremental_db
acumulador.vhd            LUT_cuadrada.m
acumulador.vhd.bak       LUT_cuadrada.vhd
acumulador_fase.qpf      LUT_seno.m
acumulador_fase.qsf      LUT_seno.vhd
acumulador_fase.qsf.bak  LUT_sierra.m
acumulador_fase.qsf     LUT_sierra.vhd
acumulador_fase.vhd      Makefile
acumulador_fase.vhd.bak  output_files
amplitud.vhd             reg32.vhd
c_program                reg32_avalon_interface.vhd
i2s_pin_model_dump.txt  reg32_avalon_interface_hw.tcl
db                       registro.vhd
embedded_system          simulation
embedded_system.qsys     sintetizador.vhd
embedded_system.sopcinfo sumador.vhd
generate.sh              sumador.vhd.bak
hps_lsw_handoff

jose@scobyLap /cygdrive/c/Reporte
$ PATH=/cygdrive/c/altera_lite/15.1/quartus/sopc_builder/bin:$PATH

jose@scobyLap /cygdrive/c/Reporte
$ ./generate.sh
binInfoHeader: Creating macro file 'hps_0.h' for module 'hps_0'

jose@scobyLap /cygdrive/c/Reporte
$

```

Figura 4.28 Generación del archivo de cabecera hps_0.h.

Con este comando ya se habrá generado el archivo de cabecera “hps_0.h” en la carpeta del proyecto de Quartus tal y como se muestra en la Figura 4.29, ahora se debe entrar a dicha carpeta y copiar y pegar dicho archivo en la carpeta jni del proyecto de eclipse.

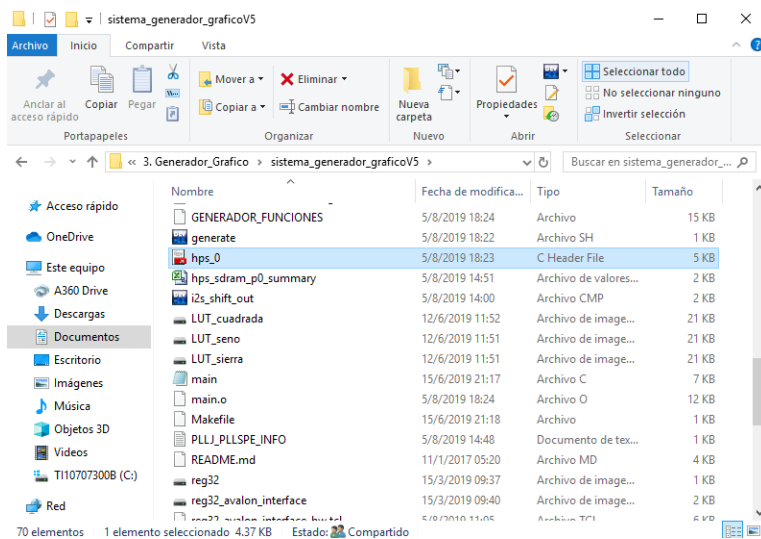


Figura 4.29 Librería hps_0.h generada a partir del Qsys del proyecto de Quartus para FPGA.

La librería hps_0.h contiene toda la información del archivo de Quartus (direcciones de memoria).

El código de C funciona a base de punteros, los cuales apuntan a las direcciones de memoria que indica el Qsys por medio de la librería hps_0.h. El código se compone de 5 funciones, las cuales son las mismas declaradas en el código JNIserver.java, en dicho código fueron declaradas para poder ser utilizadas en Java y en este programa se especifica en lenguaje C, que hace cada una de ellas.

La primera función es `init()` y es la encargada de asignar a los punteros la dirección de memoria a la cual esté apuntando, esta función solo se debe invocar una vez, ya que utiliza una función `mmap()` de la librería `mman.h` y cada que se ejecuta esta función se recorre en uno las direcciones de memoria base, si esto se repite en un número muy alto de veces se desbordará la memoria, haciendo que todo el programa falle.

En la función `enable(enable,fs)` se habilita o deshabilita el sistema en general (ADC y FFT) mandando a la respectiva dirección de memoria de esta señal de control un '1' o un '0' según sea el caso y lo que indique la variable de entrada llamada `enable`. En caso de que se habilite el sistema además se mandará al registro que está conectado a la entrada de control que selecciona la frecuencia de muestreo dicho valor el cual viene en la variable de entrada llamada `fs`. En caso de deshabilitar el sistema se mandará un ciclo en las entradas de reloj de lectura de las FIFO con el fin de resetearlas.

La función `reading(fs)` es la encargada de generar un pulso para el reloj de escritura de las memorias FIFO de cada canal habilitado del ADC, la frecuencia de esos pulsos generados se midió sacando dicha señal por el puerto de expansión del FPGA y viéndolo en un osciloscopio, dicho pulso de reloj fue de 4.5kHz. En las FIFO el reloj de escritura lleva la misma frecuencia que la frecuencia de muestreo, si se desea que la frecuencia de lectura sea siempre menor que la de escritura esto no será posible cuando se seleccione una frecuencia de muestreo menor a 4.5kHz por lo que por medio de una variable de entrada llamada `fs` se lee la frecuencia de muestreo deseada y en caso de que sea menor a 4.5kHz se aplicarán retrasos (delays) que hagan que la frecuencia de lectura generada baje a 450Hz, 45Hz, 4.5Hz y 0.45Hz. Al no ser tan sencillo utilizar una función que regrese un arreglo, se utilizó una función que devuelve una sola variable de 64 bits. Las 4 mediciones realizadas por el ADC son de 12 bits por lo cual se pueden almacenar en una variable de 16 bits y al final poder ser

concatenadas las 4 variables de 16 bits en la variable de 64 bits. Para su lectura solo bastará con desconcatenar los bits con programación de Java.

La función FFTdone() toma la lectura del registro donde está almacenada la salida de control del sistema de adquisición que indica cuando el cálculo de la FFT se terminó de calcular. Este servirá para el ciclo while que no permitirá que el código de Java avance hasta que se haya calculado la FFT para proseguir a sacar los datos de la FIFO de la FFT.

Finalmente, la función FFT() funciona de la misma forma que la función reading(fs), con la diferencia de que esta función mantiene constante su frecuencia de lectura a 4.5kHz ya que no interesa mantener una estabilidad en las frecuencias de escritura y lectura de las FIFO, ya que el reloj de lectura de la FIFO que se está controlando con esta función es la de la FFT y dicha FIFO funciona más como una RAM que como FIFO.

Una vez hecho el código en C, se deberá guardar y después modificar el código del archivo creado con anterioridad llamado makefile, con un código que permita utilizar el gcc del cygwin para compilar el programa en C y convertirlo en una librería compartida que pueda ser leída por el programa en Java.

Para correrlo se repite el proceso que se hizo con el anterior makefile. Primero se crea el target, pero esta vez el target se llamará "all" como se observa en la Figura 4.30.

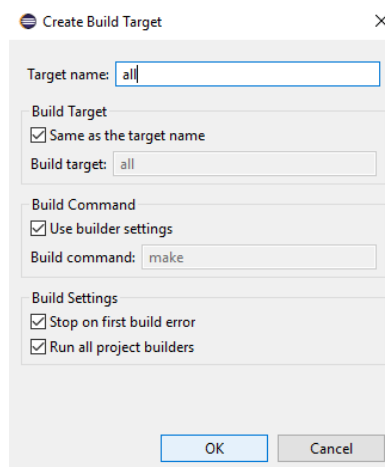


Figura 4.30 Creación del target "all".

Luego se debe seleccionar Run >> External Tools >> 1 JavahTool como se muestra en la Figura 4.31.

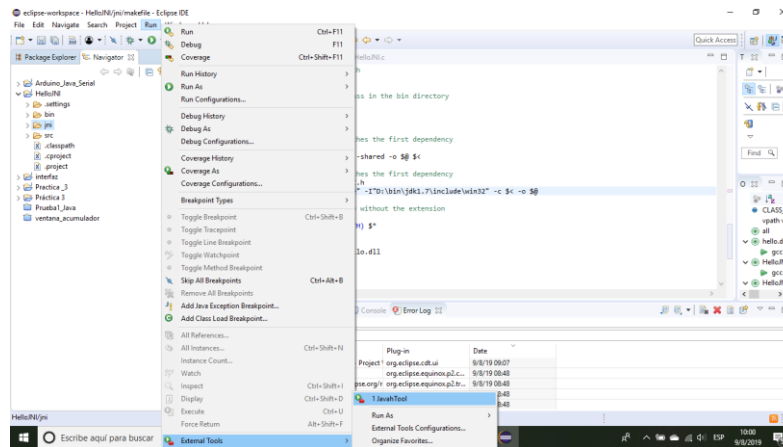


Figura 4.31 Correr Javah de nuevo.

Ahora se debe oprimir el botón derecho del ratón sobre el target llamado “all” y seleccionar “Build Targets”, tal y como se ilustra en la Figura 4.32

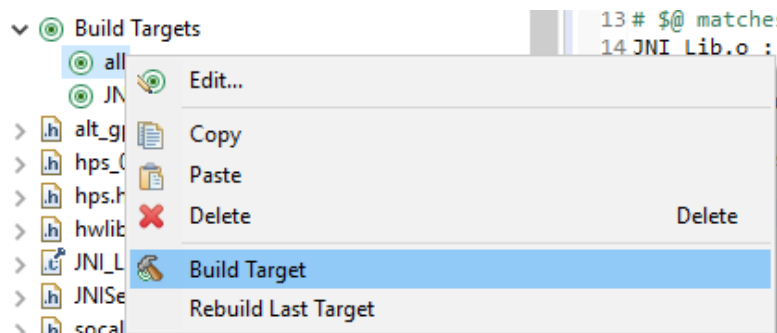


Figura 4.32 Generando “all”.

Esto creará un archivo .dll, si se logró crear sin problemas quiere decir que no hay errores en el código de C y el programa está listo para compilarse en el Linux del FPGA SoC. Pero antes de compilar los archivos en Linux Ubuntu del FPGA se debe hacer dos cosas:

- Descargar e instalar Java y gcc (compilador de C) en el sistema operativo del FPGA SoC. (Para ver cómo hacer esto revisar Apéndice I).

- Copiar archivos necesarios del proyecto de eclipse y emigrarlos al sistema operativo del FPGA SoC.

Los archivos que se requieren pasar a Linux son los que se muestran en la Figura 4.33.

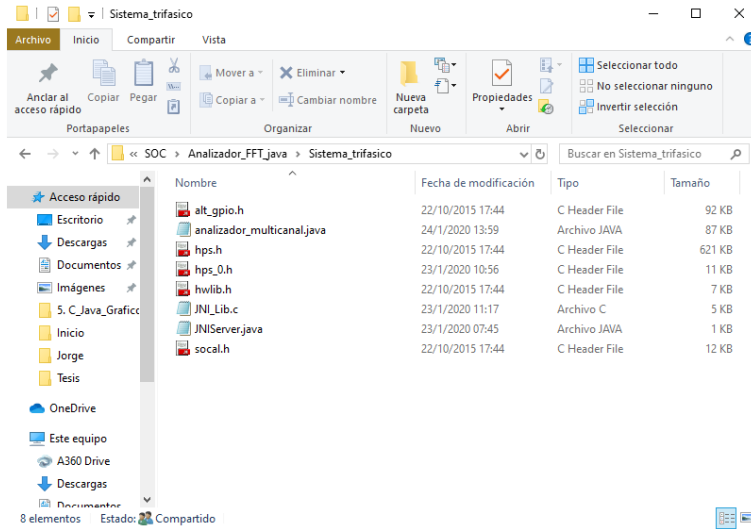


Figura 4.33 Archivos del proyecto de eclipse necesarios.

Como uno de los archivos de Java utiliza librerías que no están descargadas ni instaladas, es necesario también llevar las librerías que utiliza el código, las cuales son las que se muestran en la Figura 4.34.

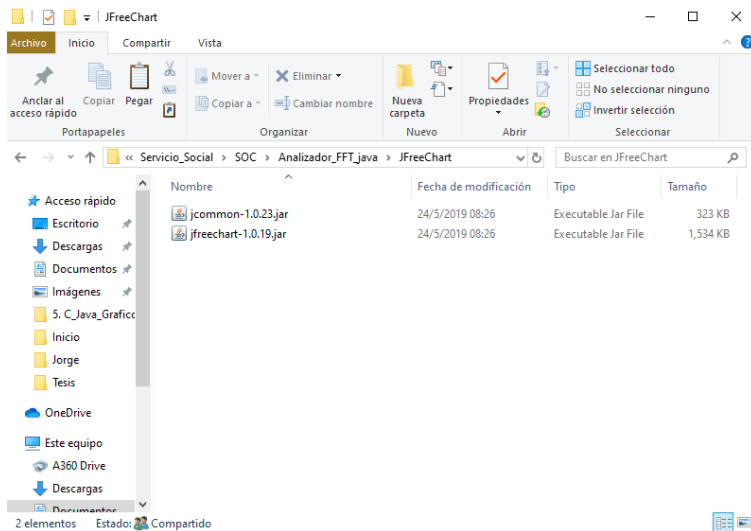


Figura 4.34 Librerías .jar que se instalarán en Linux.

Se deben colocar en carpetas y trasladar dichas carpetas a Linux. Tal y como se muestra en las Figuras 4.35 y 4.36.

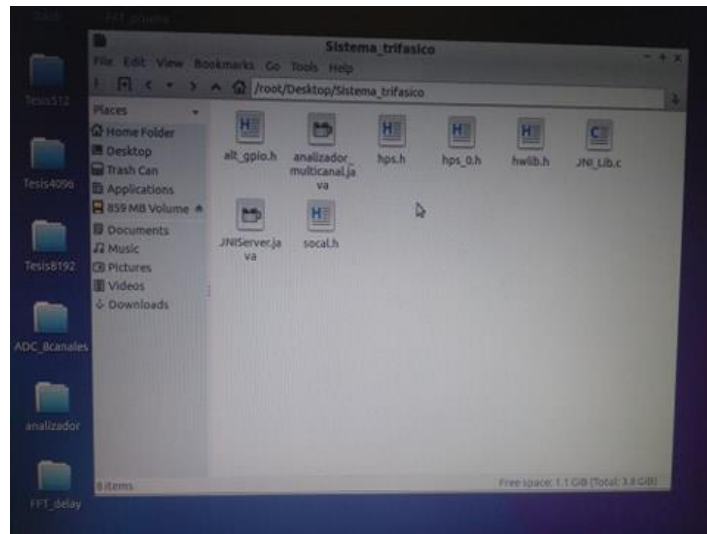


Figura 4.35 Archivos en Linux.

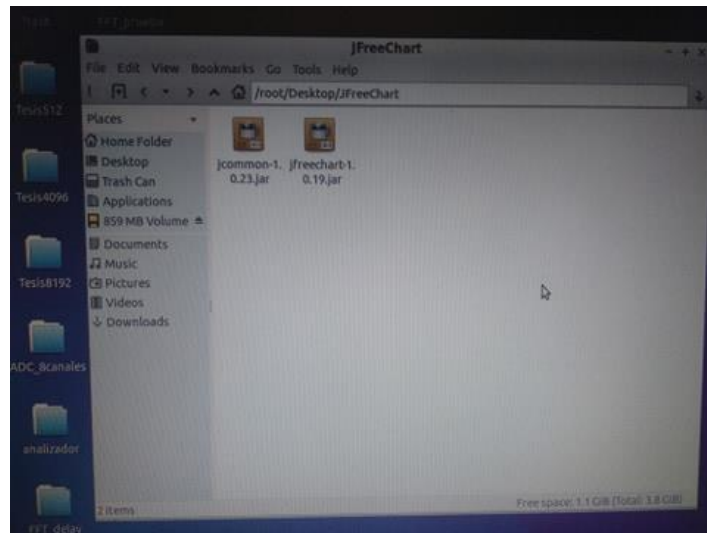


Figura 4.36 Librerías en Linux.

Se debe iniciar sesión con el root o algún usuario con privilegios de administrador (es importante esto para correr el programa, si no se hace desde un usuario con privilegios de administrador no funciona la aplicación) y abrir una terminal como se muestra en la Figura 4.37.

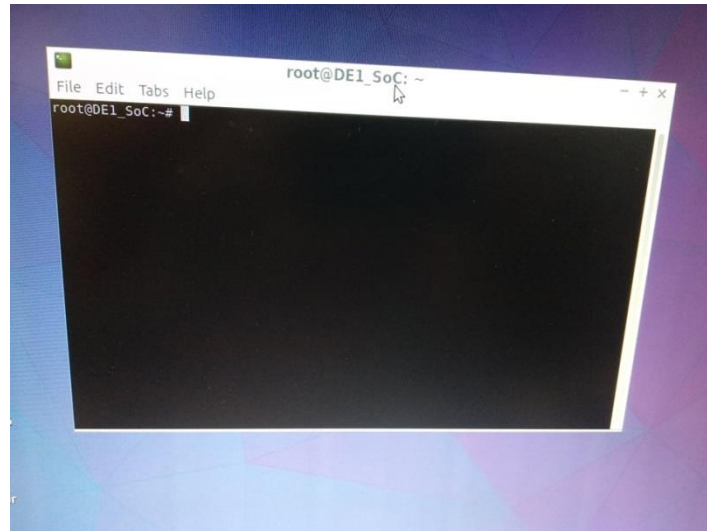


Figura 4.37 Terminal en la sesión del root.

Se debe navegar hasta donde se guardó la carpeta con los archivos. En la Figura 4.38 se observa como navegar hasta la carpeta de los archivos desde la terminal abierta con anterioridad.

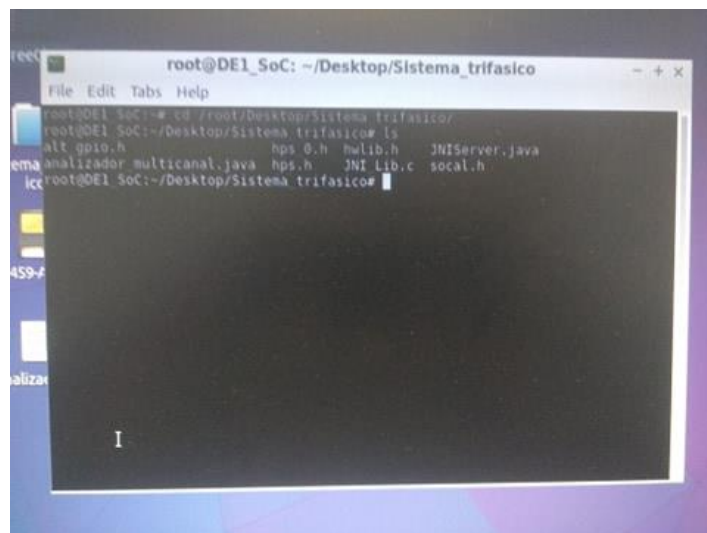


Figura 4.38 Carpeta con archivos trasladados.

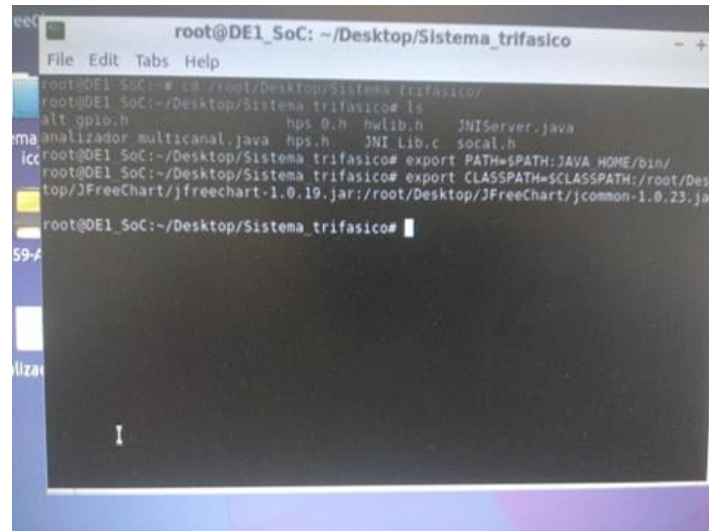
Para que Java reconozca las librerías hay que escribir dos comandos los cuales son:
[36]

```
export PATH=$PATH: /usr/local/java-current/bin
```



```
export CLASSPATH=$CLASSPATH: /libraries_location/jfreechart-1.0.18.jar:  
/libraries_location/lib/jcommon1.0.22.jar
```

En la Figura 4.39 se observa cómo se realizó el paso anterior.



```
root@DE1_SoC: ~/Desktop/Sistema_trifasico  
File Edit Tabs Help  
root@DE1_SoC: # cd /root/Desktop/Sistema_trifasico/  
root@DE1_SoC:~/Desktop/Sistema_trifasico# ls  
alt_opio.h          hps_0.h  hplib.h  JNIserver.java  
analizador_multicanal.java  hps.h    JNI Lib.c  social.h  
root@DE1_SoC:~/Desktop/Sistema_trifasico# export PATH=$PATH:JAVA_HOME/bin/  
root@DE1_SoC:~/Desktop/Sistema_trifasico# export CLASSPATH=$CLASSPATH:/root/Desk  
top/JFreeChart/jfreechart-1.0.19.jar:/root/Desktop/JFreeChart/jcommon-1.0.23.jar  
root@DE1_SoC:~/Desktop/Sistema_trifasico#
```

Figura 4.39 Inclusión de librería JFreeChart en Linux.

Luego usar el comando javac (compila los archivos .java y crea los archivos .class) con el archivo generador_funciones.java (el programa principal) [37]:

```
javac analizador_multicanal.java
```

En la Figura 4.40 se observa cómo se realizó el paso anterior.

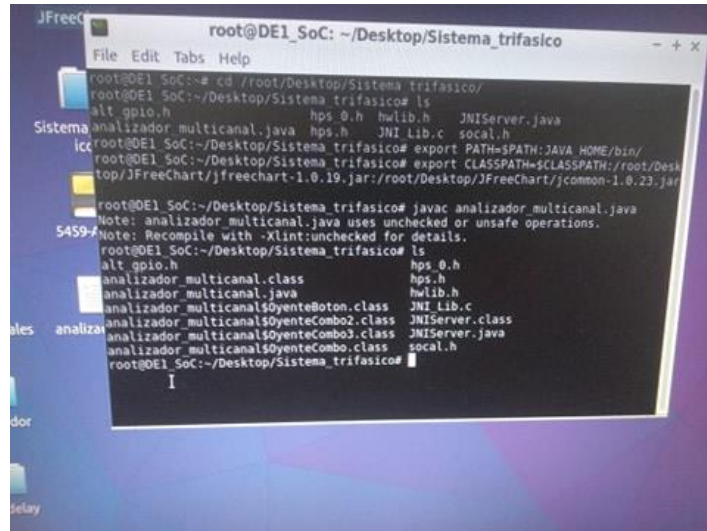


Figura 4.40 Compilación de los archivos de Java por medio del comando javac.

Luego crear el archivo de cabecera JNI_Server.h usando en comando javah con el JNI_Server.class:

```
javah JNI_Server
```

En la Figura 4.41 se observa cómo se realizó el paso anterior.

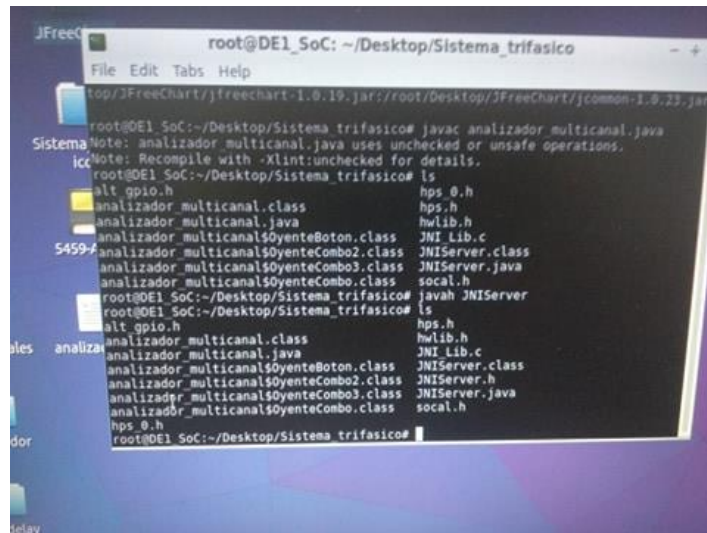


Figura 4.41 Creación del archivo de cabecera JNI_Server.h por medio del comando javah.

Para compilar el archivo de C se utiliza el siguiente comando:

```
gcc -fPIC -I"/usr/lib/jvm/java-8-openjdk-armhf/include" -  
I"/usr/lib/jvm/java-8-openjdk-armhf /include/linux" -Dsoc_cv_av -shared  
-o libJNI_Lib.so JNI_Lib.c
```

En la Figura 4.42 se observa cómo se realizó el paso anterior.

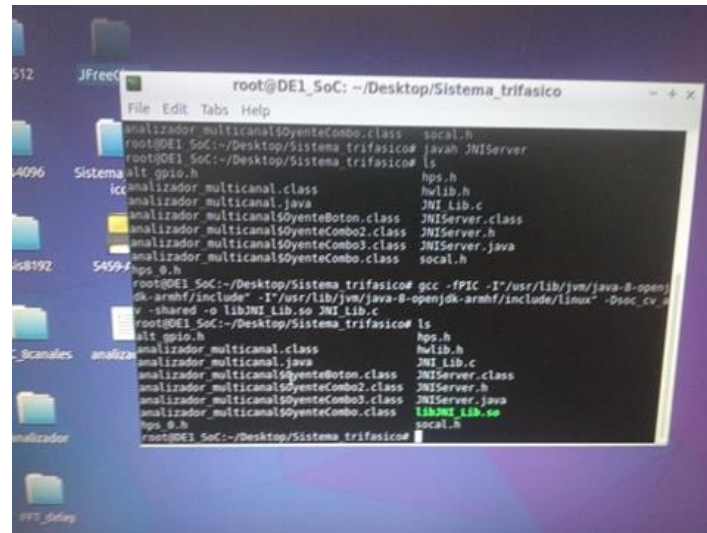


Figura 4.42 Compilación de programa en C para creación de librerías compartidas para acceder a las funciones JNI del programa desde Linux.

Y finalmente se corre el programa con el comando:

```
java -Djava.library.path=/Project_location analizador_multicanal
```

En la Figura 4.43 se observa cómo se realizó el paso anterior.

```

root@DE1_SoC: ~/Desktop/Sistema_trifasico
File Edit Tabs Help
root@DE1_SoC:~/Desktop/Sistema_trifasico# ls
alt_gpio.h                               hps.h
analizador_multicanal.class              hwlib.h
analizador_multicanal.java               JNI_Lib.c
analizador_multicanal$oyenteBoton.class  JNIserver.class
analizador_multicanal$oyenteCombo2.class JNIserver.h
analizador_multicanal$oyenteCombo3.class JNIserver.java
analizador_multicanal$oyenteCombo.class  social.h
hps_0.h
root@DE1_SoC:~/Desktop/Sistema_trifasico# gcc -fPIC -I"/usr/lib/jvm/java-8-openjdk-armhf/include" -I"/usr/lib/jvm/java-8-openjdk-armhf/include/linux" -Dsoc_cv_3
v -shared -o libJNI_Lib.so JNI_Lib.c
root@DE1_SoC:~/Desktop/Sistema_trifasico# ls
alt_gpio.h                               hps.h
analizador_multicanal.class              hwlib.h
analizador_multicanal.java               JNI_Lib.c
analizador_multicanal$oyenteBoton.class  JNIserver.class
analizador_multicanal$oyenteCombo2.class JNIserver.h
analizador_multicanal$oyenteCombo3.class JNIserver.java
analizador_multicanal$oyenteCombo.class  libJNI_Lib.so
hps_0.h                                  social.h
root@DE1_SoC:~/Desktop/Sistema_trifasico# java -Djava.library.path=/root/Desktop/Sistema_trifasico analizador_multicanal

```

Figura 4.43 Correr la interfaz gráfica desde terminal en FPGA SoC.

En la Figura 4.44 se muestra la interfaz gráfica diseñada funcionando de manera correcta en Linux Ubuntu de la tarjeta de desarrollo DE1 SoC.

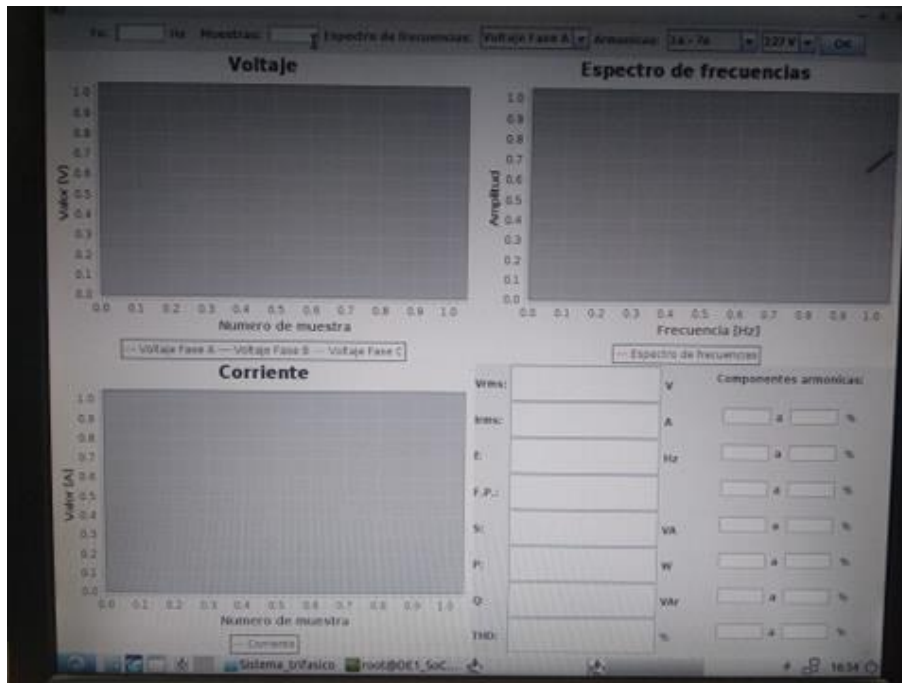


Figura 4.44 Interfaz gráfica en Linux Ubuntu de la DE1 SoC.

CAPÍTULO 5: PRUEBAS Y RESULTADOS

5.1 PRUEBA DE CONTROLADOR DEL ADC CON POTENCIÓMETRO

Para probar el funcionamiento correcto del controlador del convertidor analógico digital LTC 2308, se creó un código con el objetivo de poder cambiar la frecuencia de muestreo por medio de switches, así como el canal que se deseaba monitorear. La salida se mostrará en los LEDs con los que cuenta la tarjeta, y la lectura se obtendrá de un potenciómetro funcionando como divisor de tensión. Para poder observar la frecuencia de muestreo se sacó la señal llamada PUSH por uno de los puertos de expansión de la tarjeta (GPIO) y se observó su frecuencia (la frecuencia de esta señal es la misma que la de muestreo) en el osciloscopio.

El código sirve para convertir una entrada de 7 bits en una de 32, con el fin de poder cambiar la frecuencia de muestreo con los switches y además con ayuda de un multiplexor cuyo selector está conectado a 2 switches que servirán para seleccionar cual salida será la que se visualizará en los LEDs.

En la Figura 5.1 se puede observar el potenciómetro conectado al Header del ADC LTC2308 empotrado en la tarjeta de desarrollo DE1 SoC. Los extremos del potenciómetro se encuentran conectados a las salidas de 5V y GND mientras que el pin central del potenciómetro se encuentra conectado al canal 6 del ADC, el canal y la frecuencia de muestreo se seleccionan por medio de los switches y en los LEDs se observa la conversión realizada por el ADC. En la Figura 5.2 se puede observar la salida del reloj creado para saber la frecuencia de muestreo, al colocarle la frecuencia de muestreo máxima se observa que efectivamente como fue calculada anteriormente la frecuencia de muestreo máxima es de 124kHz.

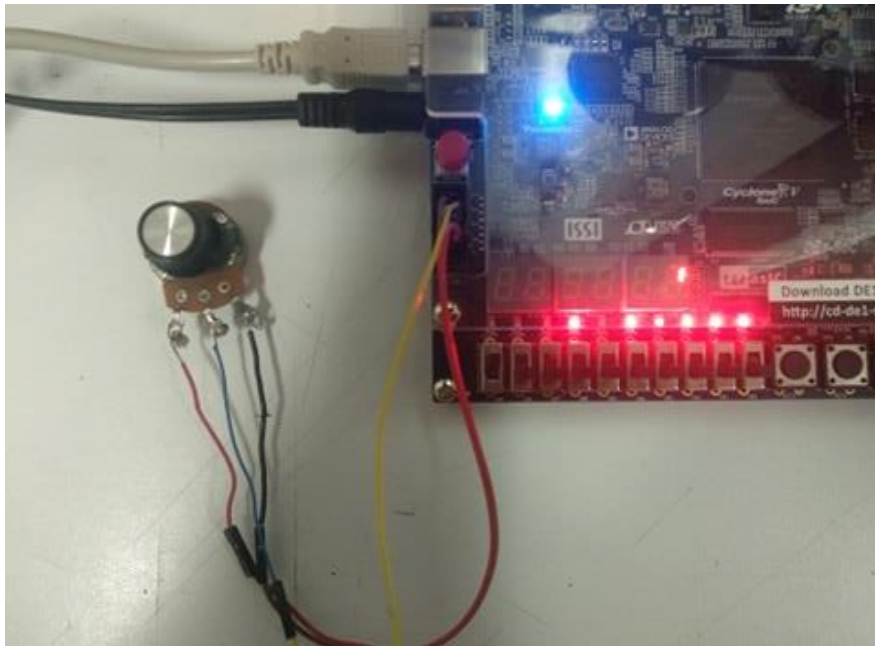


Figura 5.1 Prueba con potenciómetro.

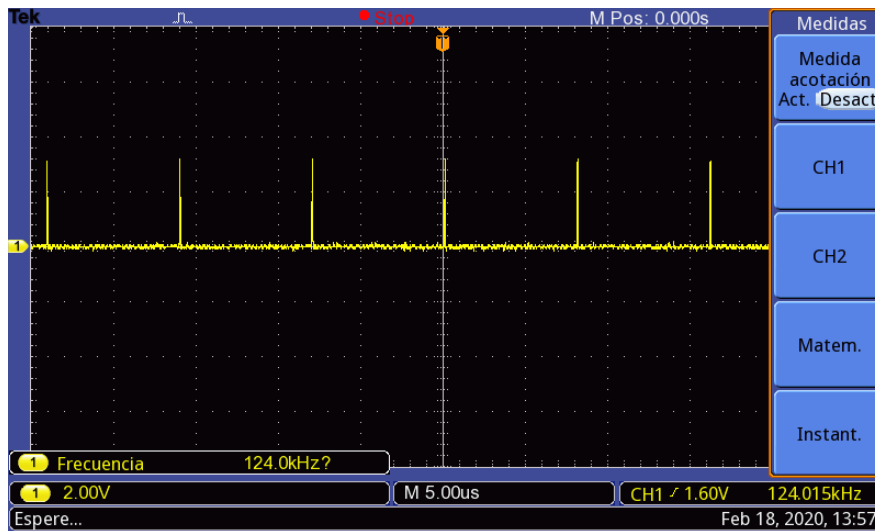


Figura 5.2 Frecuencia de muestreo observada en el osciloscopio.

5.2 PRUEBA DE FFT Y FIFO

Para las pruebas se diseñó un código en el cual permite seleccionar el canal en el que se muestrea y su respectiva FFT, las salidas del programa se mandan al puerto de expansión de la tarjeta DE1 SoC (GPIO) y poder controlar la habilitación del sistema, el canal y la frecuencia de muestreo se utilizan los switches que incorpora la tarjeta; la señal de prueba provendrá de un eliminador modificado para dar una señal entre 0 y 3.3V.

El objetivo del programa es mandar tanto las señales de salida como algunas de control a un microcontrolador Arduino Due, el cual tendrá las señales de reloj de lectura de las FIFO conectadas a dos pines, con esto generando interrupciones para que en cada interrupción se lea el dato y se guarde en un arreglo. Una vez lleno el arreglo en el Arduino Due se enviarán los datos vía serial a una terminal, donde los datos obtenidos se puedan mandar tanto a Excel como a Matlab para poder comparar los resultados.

El programa de apoyo funciona instanciando el programa principal del sistema de adquisición de señales con FFT, su frecuencia de muestreo es seleccionada con los switches, al igual que su entrada de habilitación, los relojes para sacar los datos de las FIFO se hace por medio de una señal de reloj de 5kHz generada con un acumulador de

fase. Las señales que son enviadas al Arduino Due son: 12 bits de salida de la lectura del ADC, 14 bits de salida de la FFT, la señal de reloj generada para sacar los datos de las FIFO y un bit que muestra si el sistema está habilitado o no.

El código para leer los datos, almacenarlos y mandarlos vía serial con ayuda de un Arduino Due funciona en base a interrupciones, donde cada que hay una interrupción provocada por un flanco ascendente el Arduino lee el valor de los 14 o 12 bits dependiendo si es la FFT o la lectura del ADC y lo guarda en un arreglo, cada interrupción incrementa el valor del índice del arreglo y este solo añade un valor si la entrada de habilitación está en alto, cuando se baja se resetea el código de Arduino. Una vez llenos los arreglos estos se mandan de forma serial a una terminal donde se podrán copiar los datos para su análisis.

A continuación se presenta la relación de pines entre el FPGA y el Arduino que se utilizaron:

Tabla 5.1 Relación de pines entre FPGA y Arduino Due para las pruebas.

Señal	FPGA pin	Arduino pin	Señal	FPGA pin	Arduino pin
output_fft[0]	GPIO_0[0] (PIN_AC18)	1	output[0]	GPIO_0[15] (PIN_AG17)	16
output_fft[1]	GPIO_0[1] (PIN_Y17)	2	output[1]	GPIO_0[16] (PIN_AA18)	17
output_fft[2]	GPIO_0[2] (PIN_AD17)	3	output[2]	GPIO_0[17] (PIN_AA19)	20
output_fft[3]	GPIO_0[3] (PIN_Y18)	4	output[3]	GPIO_0[18] (PIN_AE17)	21
output_fft[4]	GPIO_0[4] (PIN_AK16)	5	output[4]	GPIO_0[19] (PIN_AC20)	22
output_fft[5]	GPIO_0[5] (PIN_AK18)	6	output[5]	GPIO_0[20] (PIN_AH19)	23
output_fft[6]	GPIO_0[6] (PIN_AK19)	7	output[6]	GPIO_0[21] (PIN_AJ20)	24
output_fft[7]	GPIO_0[7] (PIN_AJ19)	8	output[7]	GPIO_0[22] (PIN_AH20)	25
output_fft[8]	GPIO_0[8] (PIN_AJ17)	9	output[8]	GPIO_0[23] (PIN_AK21)	26
output_fft[9]	GPIO_0[9] (PIN_AJ16)	10	output[9]	GPIO_0[24] (PIN_AD19)	27
output_fft[10]	GPIO_0[10] (PIN_AH18)	11	output[10]	GPIO_0[25] (PIN_AD20)	28

output_fft[11]	GPIO_0[11] (PIN_AH17)	12	output[11]	GPIO_0[26] (PIN_AE18)	29
output_fft[12]	GPIO_0[12] (PIN_AG16)	13	PUSH 2	GPIO_0[27] (PIN_AE19)	30
output_fft[13]	GPIO_0[13] (PIN_AE16)	14	ADC_en	GPIO_0[28] (PIN_AF20)	31
PUSH	GPIO_0[14] (PIN_AF16)	15			

Para comparar los resultados obtenidos se creó un programa en Matlab, donde se creara un vector con las muestras obtenidas y se le sacara la FFT, para después comparar los datos obtenidos por Matlab con los datos obtenidos por el sistema de adquisición. La comparación se hizo dato por dato y gráficamente.

En la Figura 5.3 se puede observar una señal de prueba proveniente de un eliminador modificado para que rectifique una señal sinusoidal de 60Hz. En la Figura 5.4 se aprecian los componentes para las pruebas los cuales son: Una laptop con terminal serial, un FPGA y un Arduino Due, la señal mostrada en la Figura 5.3 ingresa al ADC, el cual con ayuda del FPGA adquiere los datos en la FIFO, al mismo obtiene la FFT y las manda los resultados (dato digitalizado y FFT) de manera paralela al Arduino Due el cual se encarga de mandar via serial los datos adquiridos para su observación en la computadora. En la Figura 5.5 se muestran los datos recibidos en la terminal serial, primero se mandan separados por un espacio para poder utilizarlos en Matlab, y luego se mandan separados por un retorno de línea para poder utilizarlos en Excel. En la Figura 5.6 se observan los datos adquiridos por la terminal serial pero graficados en Excel. En la Figura 5.7 se muestran los datos digitalizados graficados en Matlab, así como la gráfica de la FFT obtenida de esos mismos datos en Matlab con el fin de compararlos con los obtenidos por el FPGA. En la Figura 5.8 se muestran la comparación de la FFT obtenida con Matlab contra los obtenidos por el sistema con el fin de encontrar el porcentaje de error del sistema.

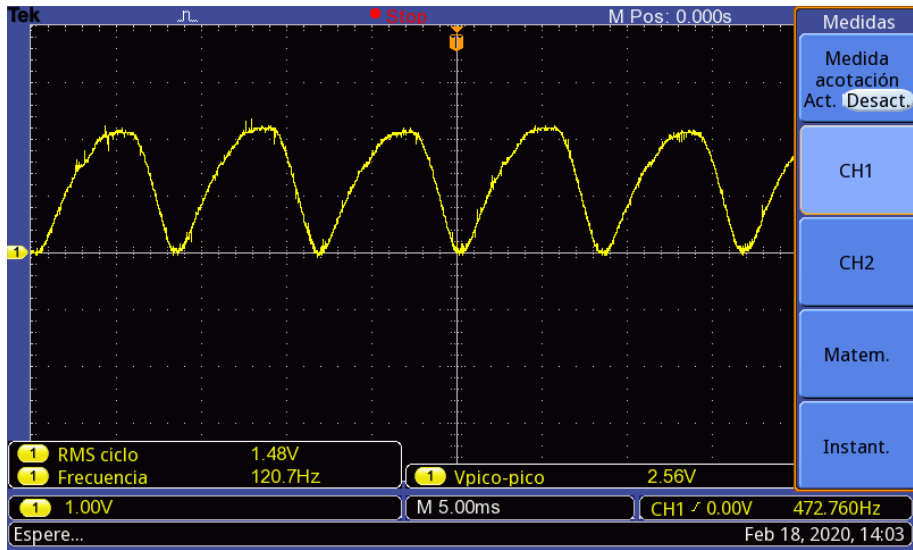


Figura 5.3 Señal de prueba.

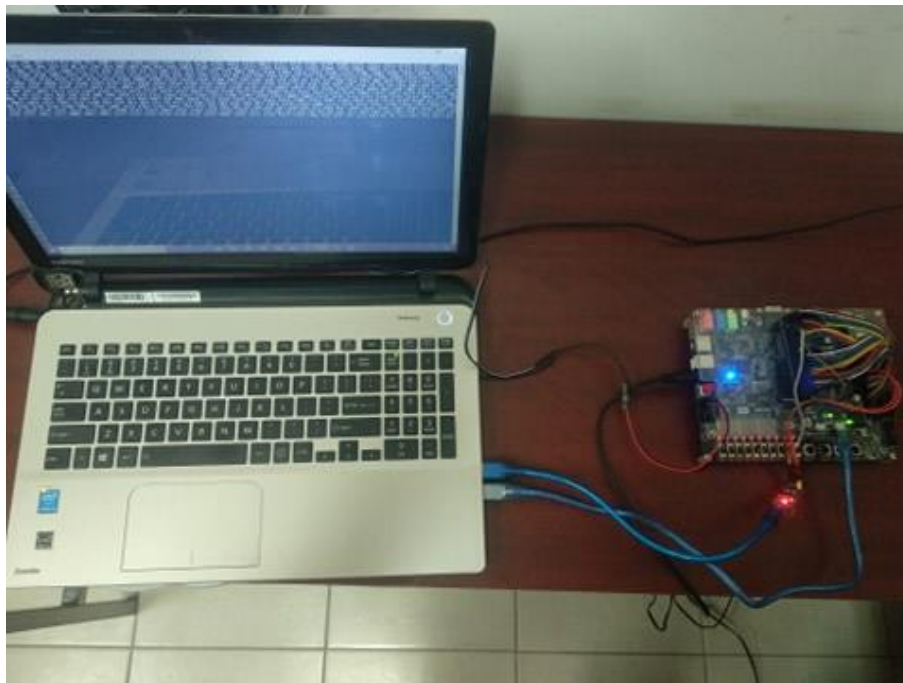


Figura 5.4 Sistema de pruebas.

1	2	3	4	5	6	7	8	9	10	11	12	13
1370	31	36	69	507	43	14	18	108	23	12	0	39
1	2	3	4	5	6	7	8	9	10	11	12	13
1373	30	41	67	498	33	12	17	105	19	9	6	36

Figura 5.8 Comparación de datos de la FFT obtenida con el sistema contra los datos obtenidos por el algoritmo de Matlab.

Comparando los resultados obtenidos en la prueba con una señal real, la FFT tiene un error de 1.06% con respecto al algoritmo que realiza Matlab, lo cual lo convierte en un sistema altamente confiable.

5.3 PRUEBA EN SIMULADOR DEL CIRCUITO DE ACONDICIONAMIENTO DE SEÑALES

Se realizó una prueba para cada caso analizado con los valores obtenidos analíticamente y se probaron en simulación con el software Multisim [38].

El circuito que se utilizó se ilustra en la Figura 5.9. Lo único que se hizo fue variar el valor de las resistencias según sea el caso de la prueba.

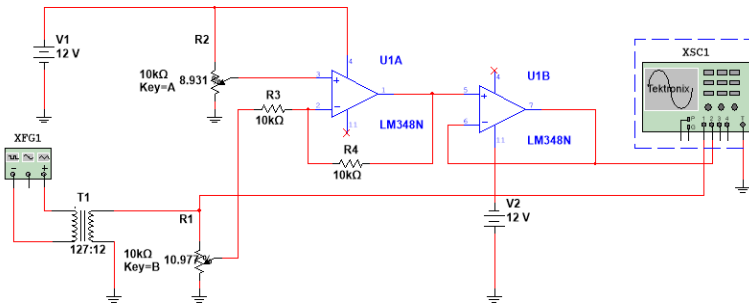


Figura 5.9 Circuito de pruebas.

Para la prueba de voltaje fase A, B o C a 127 V, se utilizó un generador de funciones para crear la onda de 127 V_{rms}. Los parámetros de las resistencias se ajustaron de la siguiente forma:

$$R_{\%1} = 10.977\% \quad (123)$$

$$R_{\%2} = 8.931\% \quad (124)$$

$$R_3 = 10k\Omega \quad (125)$$

$$R_4 = 10k\Omega \quad (126)$$

Para observar los resultados se conectó un osciloscopio de la marca Tektronix, las señales que se muestran son la del voltaje en el lado del secundario del transformador y la señal de salida del circuito (entrada al ADC). El osciloscopio se ajustó para que mostrara las frecuencias de ambas señales, sus voltajes pico-pico y el voltaje medio de la señal de salida con el fin de poder observar el offset.

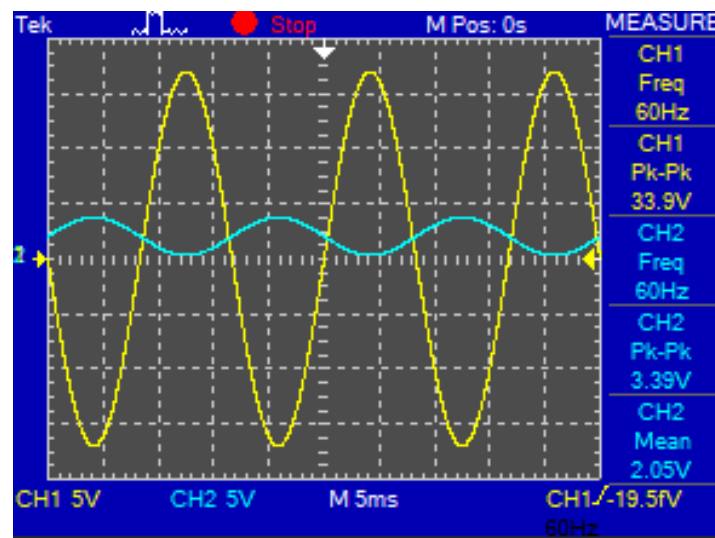


Figura 5.10 Prueba del circuito para medición de voltajes fase A, B o C a 127 V.

Los resultados de esta prueba se pueden observar en la Figura 5.10 y fueron los esperados obteniendo un offset de 2.05 V y una onda con voltaje pico-pico de 3.39 V.

Para la prueba de voltaje fase A, B o C a 220 V, se utilizó un generador de funciones para crear la onda de 220 V_{rms}. Los parámetros de las resistencias se ajustaron igual que en las ecuaciones 123, 124, 125 y 126.

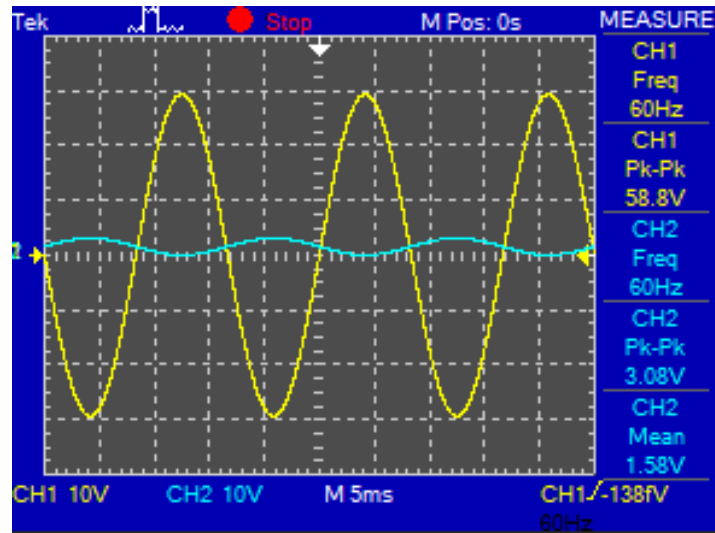


Figura 5.11 Prueba del circuito para medición de voltajes fase A, B o C a 220 V.

Los resultados de esta prueba se pueden observar en la Figura 5.11 y fueron los esperados obteniendo un offset de 1.58 V y una onda con voltaje pico-pico de 3.08 V.

Para la prueba de corriente a 127 V, se utilizó un generador de funciones para crear la onda de 35 V_{RMS}. Y por medio de un transformador con relación 100 a 2.784 reducir la onda a una de 2.76 V_{pp}. Los parámetros de las resistencias se ajustaron de la siguiente forma:

$$R_{\%1} = 16.951\% \quad (127)$$

$$R_{\%2} = 1.747\% \quad (128)$$

$$R_3 = 10k\Omega \quad (129)$$

$$R_4 = 100k\Omega \quad (130)$$

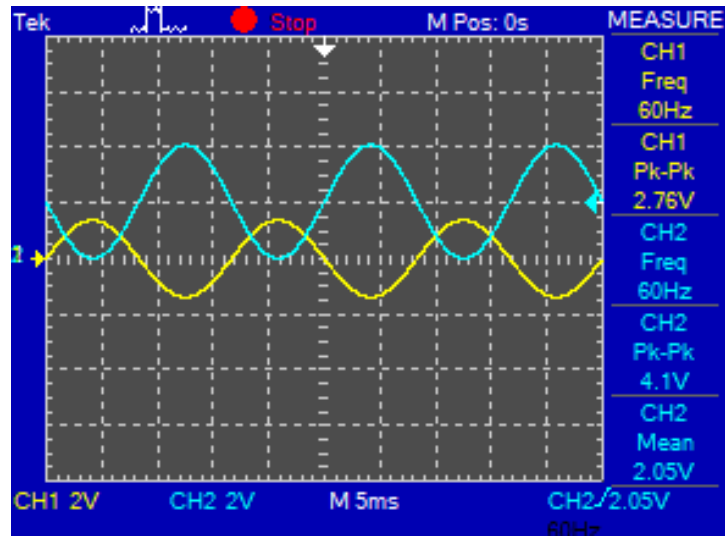


Figura 5.12 Prueba del circuito para medición de corrientes a 127 V.

Los resultados de esta prueba se pueden observar en la Figura 5.12 y fueron los esperados obteniendo un offset de 2.05 V y una onda con voltaje pico-pico de 4.1 V. Pudiendo así meter ondas de corriente con I_{RMS} de 0 a 35 A.

Para la prueba de corriente a 220 V, se utilizó un generador de funciones para crear la onda de 38.141 V_{RMS} . Y por medio de un transformador con relación 100 a 2.784 reducir la onda a una de 3 V_{pp} . Los parámetros de las resistencias se ajustaron de la al igual que en las ecuaciones 127, 128, 129 y 130.

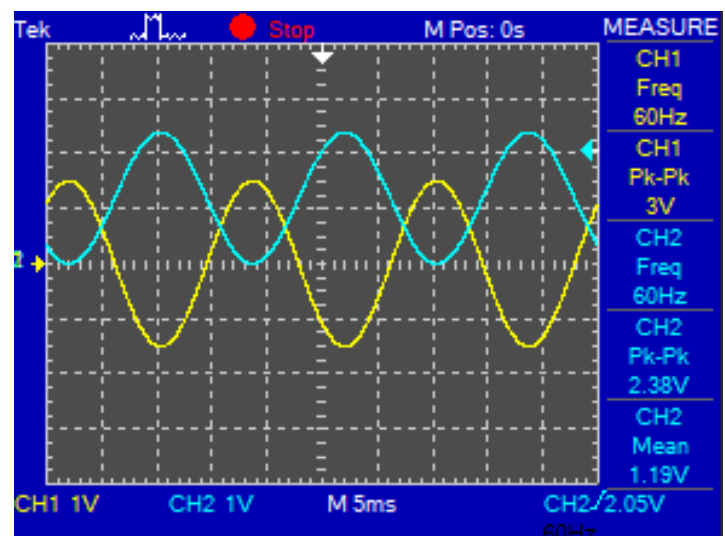


Figura 5.13 Prueba del circuito para medición de corrientes a 220 V.

Los resultados de esta prueba se pueden observar en la Figura 5.13 y fueron los esperados obteniendo un offset de 1.19 V y una onda con voltaje pico-pico de 2.38 V . Pudiendo así meter ondas de corriente con I_{RMS} de 0 a 38.141 A .

5.4 SISTEMA EN GABINETE

El circuito en el gabinete y el gabinete se pueden apreciar en las Figuras 5.14, 5.15, 5.16, 5.17.



Figura 5.14 Circuito implementado en el gabinete junto con el FPGA y los transformadores de medición.



Figura 5.15 Vista superior del gabinete.



Figura 5.16 Vista trasera del gabinete.



Figura 5.17 Vista frontal del gabinete.

5.5 PRUEBA DE ENLACE HPS-FPGA

En la Figura 5.18 se puede observar al FPGA funcionando con Linux Ubuntu, al FPGA están conectados además un teclado y un mouse para poder controlar el sistema operativo. En la Figura 5.19 se observa el escritorio de la versión de Linux Ubuntu diseñada para sistemas empotrados como SoC FPGA.



Figura 5.18 Linux Ubuntu funcionando con el FPGA.

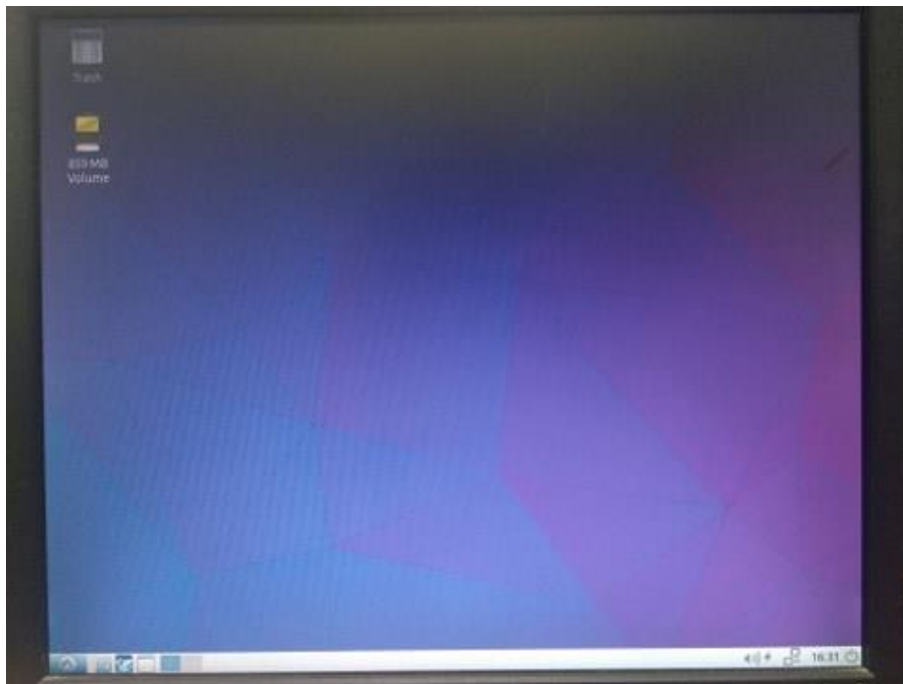


Figura 5.19 Escritorio de Linux Ubuntu para DE1 SoC.

5.6 PRUEBAS DE INTERFAZ GRÁFICA DE USUARIO

Con el fin de probar la interfaz gráfica sin conexión a los punteros de memoria, se diseñó un banco de pruebas, el cual consisten en introducir manualmente las señales que provienen del FPGA, en total las señales (en tiempo continuo) para probar la interfaz gráfica fueron:

$$v_A(t) = 220 \sqrt{\frac{2}{3}} \cos(2\pi ft) \quad (131)$$

$$v_B(t) = 220 \sqrt{\frac{2}{3}} \cos(2\pi ft + 120^\circ) \quad (132)$$

$$v_C(t) = 220 \sqrt{\frac{2}{3}} \cos(2\pi ft - 120^\circ) \quad (133)$$

$$i_A(t) = 5 \cos(2\pi ft + 30^\circ) \quad (134)$$

Para poder pasar las ecuaciones 131, 132, 133 y 134 a tiempo discreto es necesaria solamente la frecuencia de muestreo, dato que es requerido para ejecutar el programa y puede ser tomado de ahí. Las señales que se llenarán automáticamente para probar la interfaz gráfica en tiempo discreto serán:

$$v_A(n) = 220 \sqrt{\frac{2}{3}} \cos(2\pi f \frac{n}{f_s}) \quad (135)$$

$$v_B(n) = 220 \sqrt{\frac{2}{3}} \cos(2\pi f \frac{n}{f_s} + 120^\circ) \quad (136)$$

$$v_C(n) = 220 \sqrt{\frac{2}{3}} \cos(2\pi f \frac{n}{f_s} - 120^\circ) \quad (137)$$

$$i_A(n) = 5 \cos(2\pi f \frac{n}{f_s} + 30^\circ) \quad (138)$$

Para llenar también las gráficas del espectro de frecuencia se colocaron simples funciones matemáticas:

$$E_{v_A}(n) = 185 - \frac{1536}{2125} n \quad (139)$$

$$E_{v_B}(n) = 184 - 33 \ln n \quad (140)$$

$$E_{v_C}(n) = \frac{185}{n} - \frac{1536}{2125} \quad (141)$$

$$E_{i_A}(n) = 184 \quad (142)$$

Con dichas gráficas se obtuvieron resultados positivos y se lograron afinar detalles y detectar errores para obtener la versión final del código.

En la Figuras 5.20, 5.21, 5.22 y 5.23 se aprecia cómo va funcionando la interfaz gráfica con las señales de prueba generadas, la diferencia entre cada una es el espectro de frecuencias que se muestra.

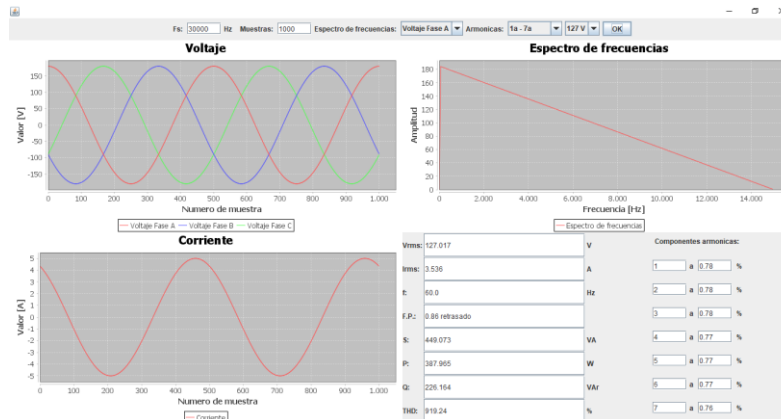


Figura 5.20 Prueba de programación en Java 1.

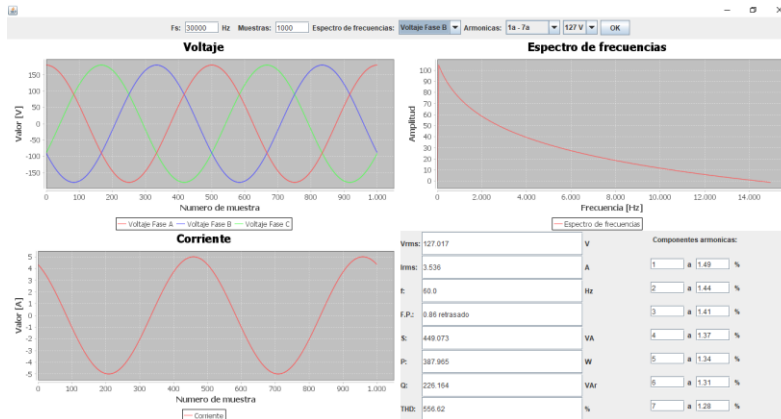


Figura 5.21 Prueba de programación en Java 2.

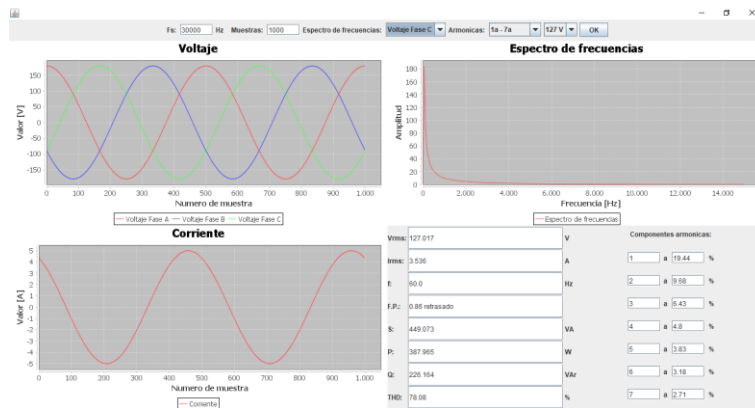


Figura 5.22 Prueba de programación en Java 3.

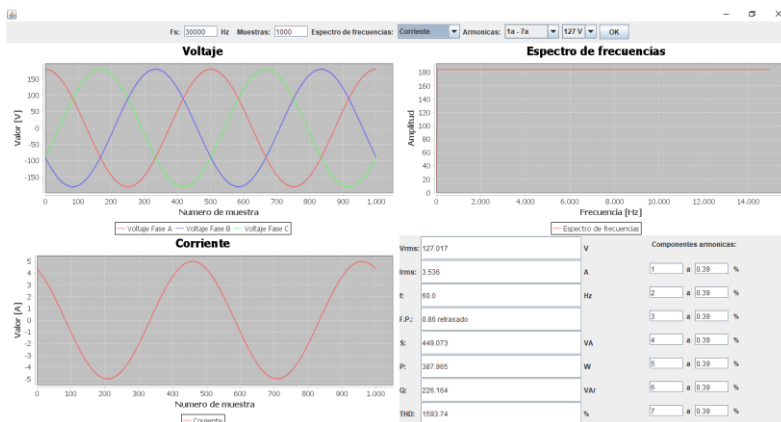


Figura 5.23 Prueba de programación en Java 4.

5.7 RESULTADOS

Para probar el sistema se decidió compararlo contra un equipo comercial de la marca Dranetz, tanto el equipo como el sistema se colocaron en el centro de carga de las instalaciones del Taller Multidisciplinario Básico (TAMULBA) de la Universidad Autónoma del Estado de Morelos (UAEM), se decidió probar en ese punto ya que ahí se concentran grandes cargas de corriente eléctrica, lo que logra deformaciones en los voltajes de línea a neutro, además de que en este punto se pueden medir tres fases de voltaje. Para medir la corriente se utilizó un motor de bomba para agua de 250 kW conectado a la red de 127 V.

En la Figura 5.24 se puede observar el PQA (Power Quality Analyzer) de baja tensión Power Visa de la marca Dranetz, equipo con valor comercial de \$111,000 MXN aproximadamente. Dicho equipo fungirá como equipo contra el que se compara el sistema implementado con tecnología SoC FPGA. En la Figura 5.25 se puede observar el lugar donde se realizaron las mediciones en el centro de carga, todo se realizó donde se encuentran los fusibles, al no contar con un neutro desnudo, la conexión del neutro se realizó en la tierra física, y los caimanos de fase se conectaron a cada una de las respectivas fases. En la Figura 5.26 se observa el motor de bomba que se utilizó para medir la corriente, se tuvo que escoger el motor debido a que el grosor de los cables del centro de carga era demasiado grueso para el transformador de corriente con el que cuenta el sistema. En la Figura 5.27 se pudo observar el sistema basado en SoC FPGA y el equipo Dranetz conectados para comparar resultados y en la Figura 5.28 se aprecia todos los equipos utilizados para las pruebas, así como las conexiones que se realizaron en el centro de carga.

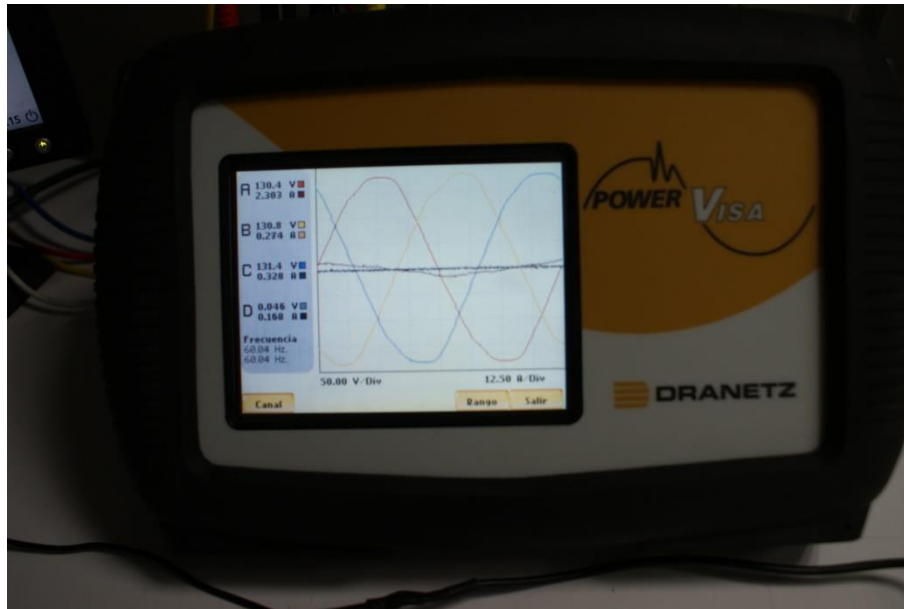


Figura 5.24 Equipo de prueba Dranetz Power Visa



Figura 5.25 Centro de carga del TAMULBA

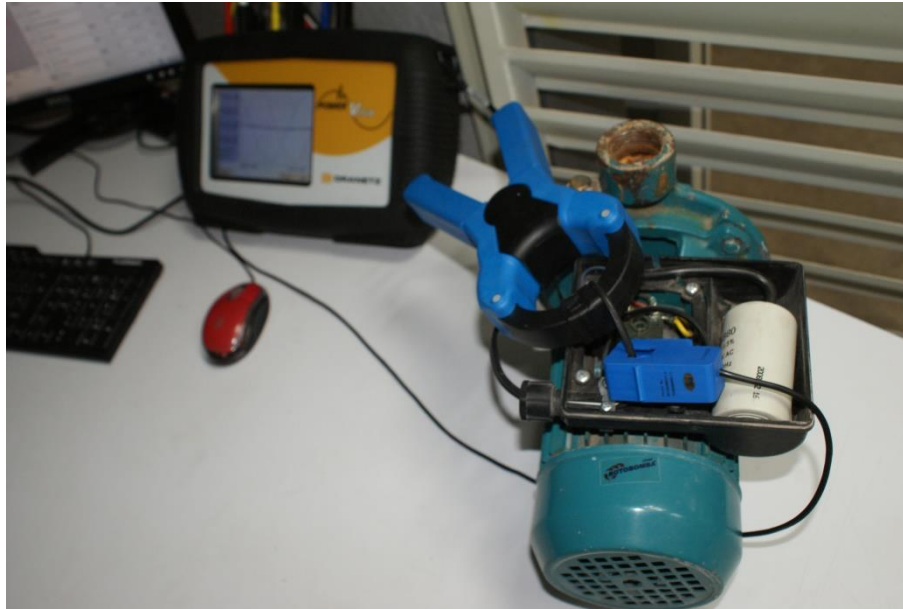


Figura 5.26 Motor de bomba de agua



Figura 5.27 Sistema instalado para las pruebas

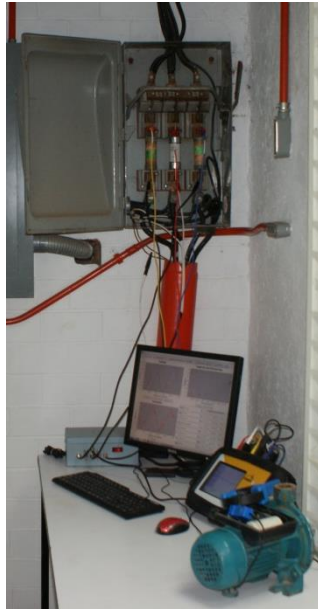


Figura 5.28 Conexiones para la prueba

En la Figura 5.29 se pueden observar los resultados donde se obtienen las tres señales de voltaje y la de señal de corriente. En esta ilustración se puede ver también el espectro de frecuencias de la corriente obtenido, y los parámetros calculados con los datos adquiridos, los cuales son muy similares a los obtenidos por el PQA de la Figura 5.24.

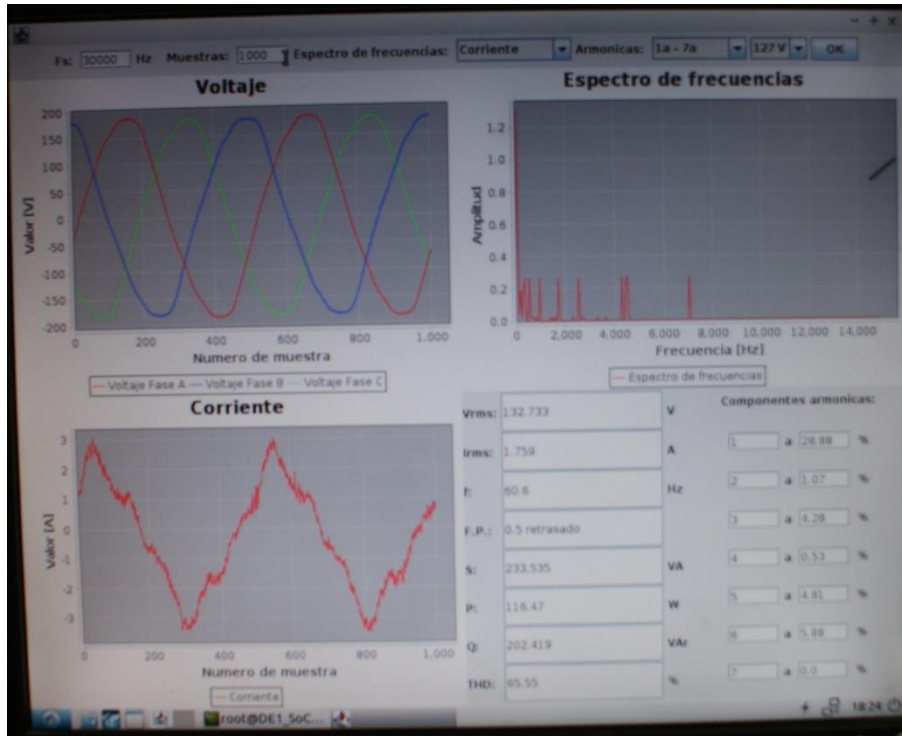


Figura 5.29 Resultados obtenidos con el sistema

5.8 CONCLUSIONES Y RECOMENDACIONES

En este trabajo se realizó un sistema de medición de la calidad de la energía utilizando tecnología SoC FPGA. Se diseñaron arquitecturas hardware para adquisición y procesamiento de señales eléctricas, así como también se realizó circuitería analógica para el acondicionamiento de las señales eléctricas. Se realizó el enlace hardware-software y se controló el sistema por medio de una interfaz gráfica de usuario. El sistema generado se colocó ante pruebas de confiabilidad y estabilidad entre las que destacan:

- Prueba de estabilidad al dejar el equipo conectado en un lapso grande de tiempo.
- Sistema no dependiente de sistema operativo externo.
- Calibración y comparación contra valores conocidos.

Para probar el funcionamiento óptimo del sistema se tuvo que realizar una prueba en comparación con un medidor de la calidad de la energía comercial y realizar mediciones en un centro de carga de baja tensión. En dicha prueba se debió revisar:

1. Comparación en cálculos de parámetros eléctricos.
2. Precisión en el espectro de frecuencias.
3. Saturación en canales analógicos.
4. Tiempos de respuesta.
5. Comparación de señales adquiridas.

La prueba debe realizarse para comprobar la capacidad del sistema.

5.8.1 CONCLUSIONES

Con lo realizado a través de este trabajo se llegó a las siguientes conclusiones:

- SoC FPGA es una variante para la realización de sistemas autónomos y una opción confiable por sus características de FPGA y amigable para el usuario por sus características de microprocesador.
- Cuando se tienen sistemas de alto costo es bueno tener alternativas de bajo costo, SoC FPGA ofrece esta solución.
- En enlace entre componentes por medio de localidades de memoria hace que el sistema sea menos susceptible a fallas, ya que no hay comunicación entre ambos chips por medio de protocolos, o cables que puedan provocar alguna falla entre la comunicación.
- Además de lograr desarrollar un sistema de monitoreo, se logró controlar entradas y salidas de una arquitectura FPGA, por lo que este tipo de sistema es aplicable para todas aquellas soluciones que requieran un FPGA
- Como sistema de monitoreo se tiene una alta velocidad en frecuencia de muestreo ya que en este sistema se multiplexaron 4 canales y aun así se consiguió una frecuencia de muestreo de hasta 124kHz. En una segunda versión del sistema se pueden incluir otros 2 canales para muestrear 3 voltajes y 3 corrientes, con esto mejorando el sistema, la frecuencia de muestreo caería a

82kHz pero aun así sigue siendo lo bastante alta como para detectar con el espectro de frecuencias componentes armónicas del orden de las milésimas.

- Dado que el sistema no depende de una computadora para realizar adquisición, procesamiento, o visualización, el sistema es considerado completamente autónomo.
- La disponibilidad de tecnología de bajo costo y de librerías de código abierto, facilitan el desarrollo de sistemas para procesar señales utilizando algoritmos computacionales complejos.
- En este trabajo se abrió la puerta para una infinidad de proyectos utilizando SoC FPGA, cualquier aplicación que un FPGA pueda hacer tiene un lugar para convertirse en un sistema SoC FPGA.

5.8.2 RECOMENDACIONES

Se recomienda continuar con sistemas SoC FPGA tomando en cuenta los siguientes puntos:

- En este trabajo por la disponibilidad del equipo en los laboratorios de la facultad se utilizó una tarjeta DE1 SoC, pero esta no es la más potente del mercado, cuenta con procesador ARM Cortex A9, un procesador bueno pero existen tarjetas de desarrollo Intel con procesadores ARM Cortex A53 que son capaces de levantar un sistema operativo con más recursos. Sería interesante ver el comportamiento de un sistema con un procesador tan poderoso y un FPGA, claro que el costo de una tarjeta así sería un problema.
- Este tipo de sistemas no es un campo explorado mucho así que no hay mucha información sobre desarrollo de este tipo de sistemas, algo clave que ayudo al desarrollo de estos sistemas fue no globalizar las búsquedas si no buscar detalle por detalle las cosas, para así después globalizarlas y crear un sistema con estas características.
- En cuanto al medidor de la calidad de la energía es recomendable añadir los canales para la medición de las corrientes que falta, para así poder obtener todos los parámetros eléctricos para cada uno de las fases, y poder tener criterios

suficientes para determinar si se cumple o no con las normas STD IEEE 519-1992; STD IEEE 1159; CFE L0000-45 o el código de red.

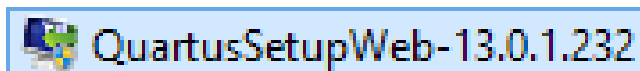
APÉNDICES

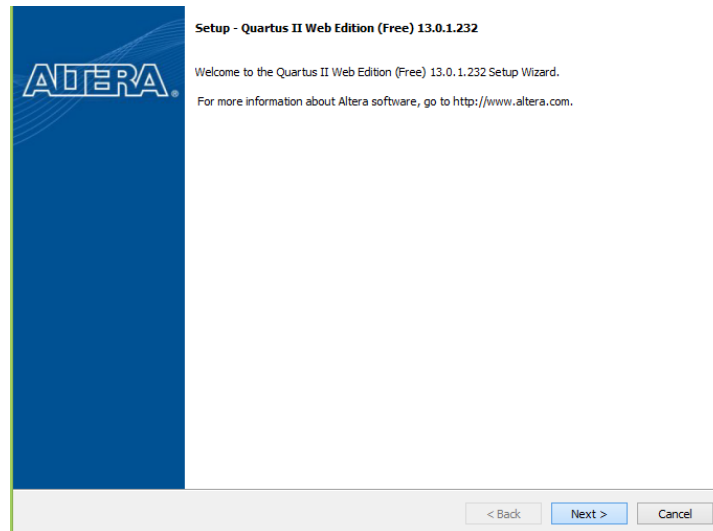
A. INSTALACIÓN DE QUARTUS

Para instalar este software es necesario descargarlo de la página web de Altera/Intel [38]. Se debe oprimir el botón «Download Software Web Edition – Free» para descargar la versión gratuita del software Quartus II (Esta versión puede ser renovada ilimitadamente sin coste):

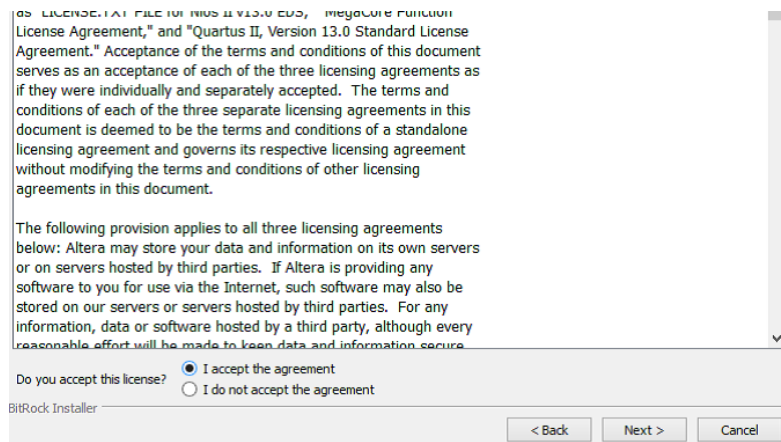


Una vez descargado el software se debe ejecutar el archivo Setup donde iniciara un programa:

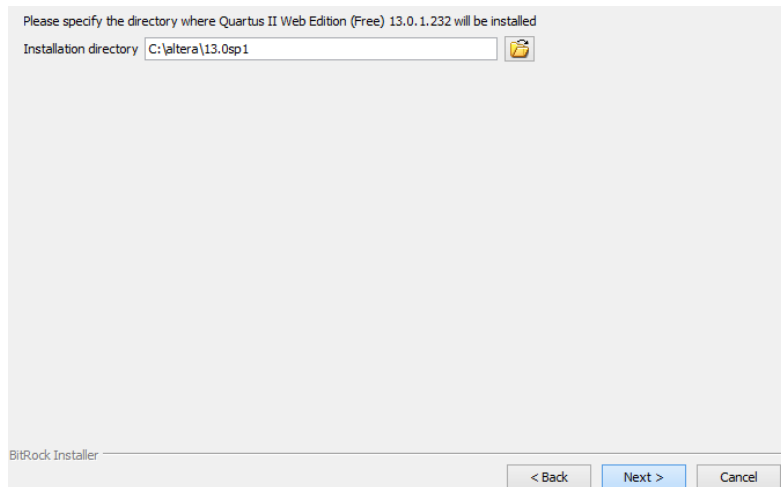




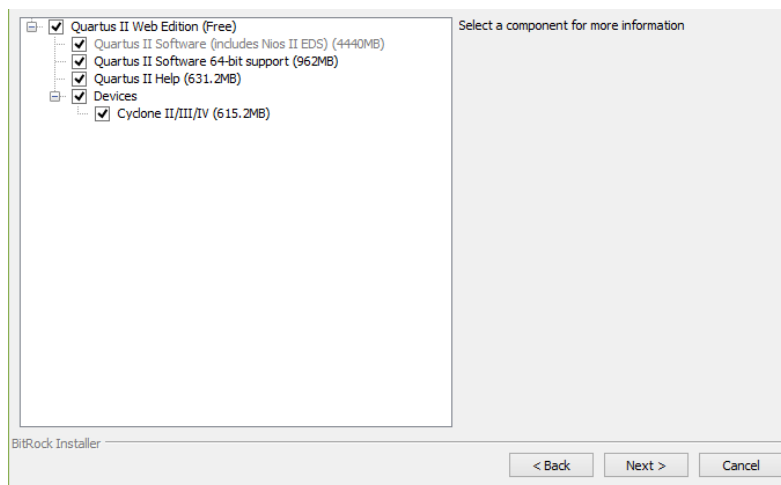
Se procede a aceptar la licencia de uso de Software:



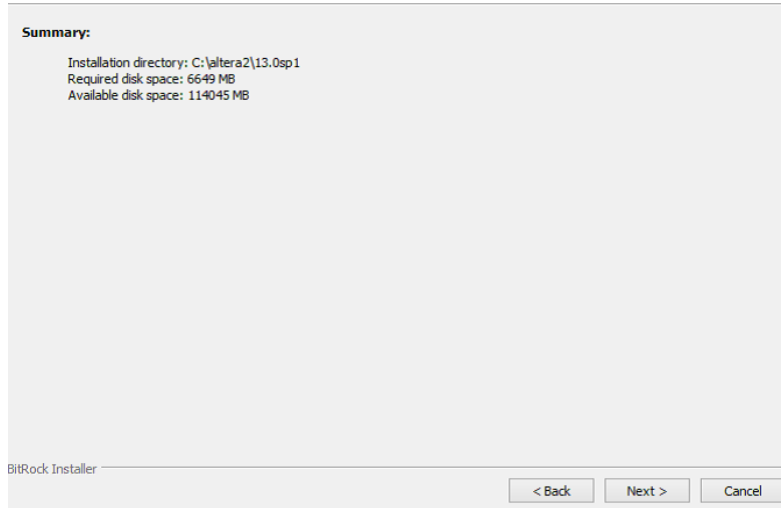
Después se debe elegir donde se guardaran los documentos en el disco duro:



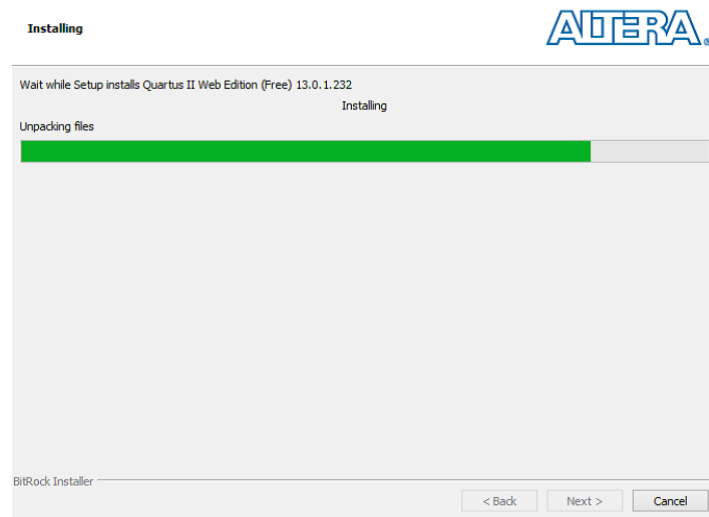
Lo siguiente será asegurarse de instalar el software completo:



Después aparecerá donde se guardara el programa y los requisitos que debe tener su PC para la instalación:

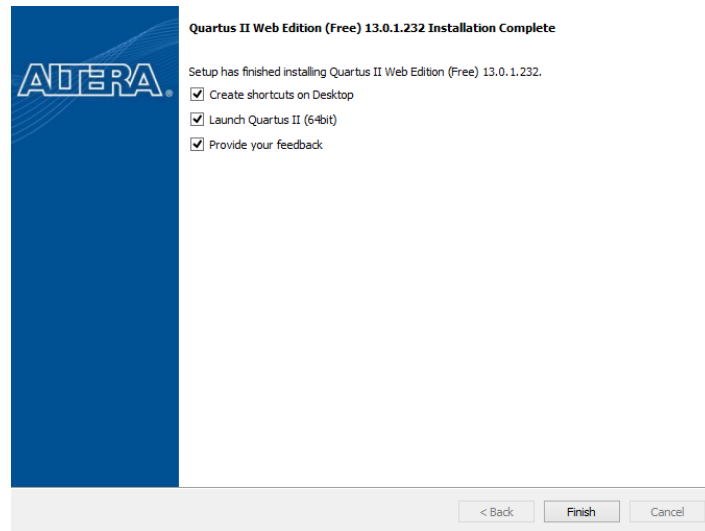


Al dar siguiente el programa comenzara a instalarse en su PC. Nota: El programa demorara varios minutos dependiendo la velocidad de su PC debido a que este programa maneja mucha información.



Al finalizar la instalación hay que marcar los iconos que aparecen:

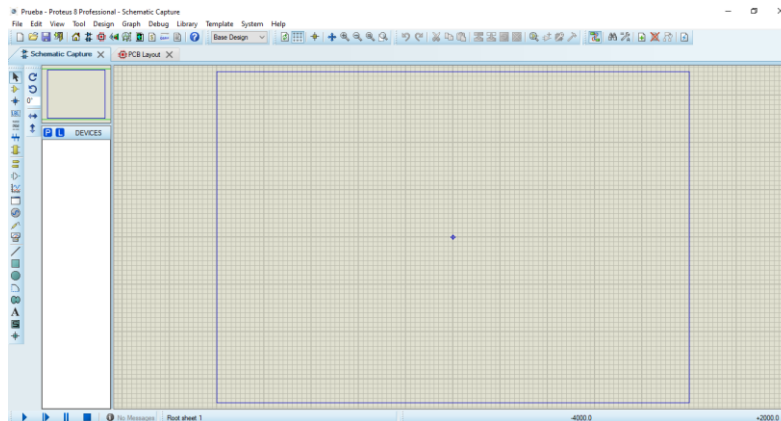
- Crear un acceso directo en el escritorio (Opcional)
- Iniciar Quartus II a 64Bits
- Proporcionar una Retroalimentación



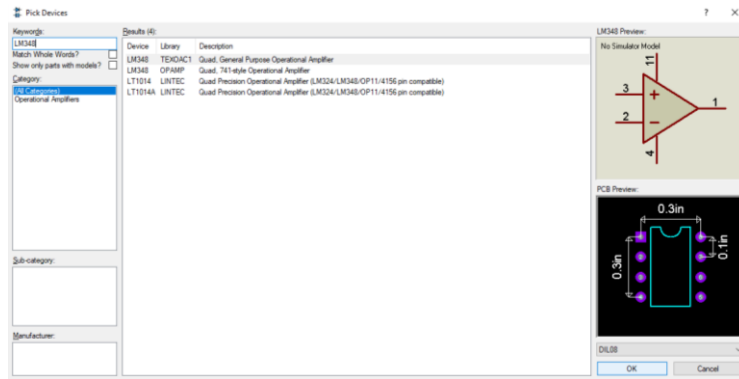
B. DISEÑO DE PCB EN PROTEUS

Para diseñar el PCB se utiliza el software Proteus. [13]

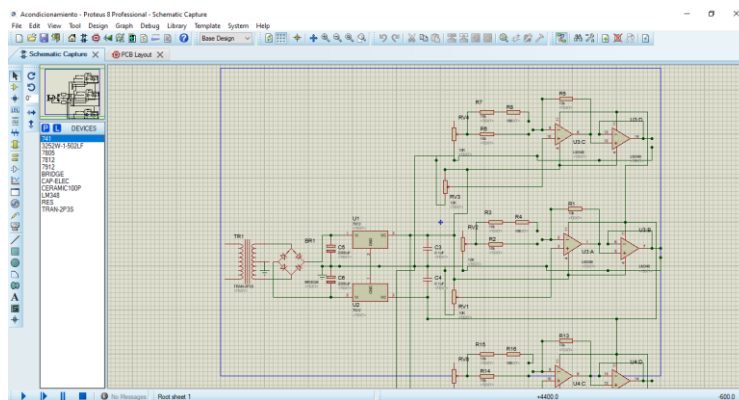
En Proteus se debe abrir un nuevo proyecto:



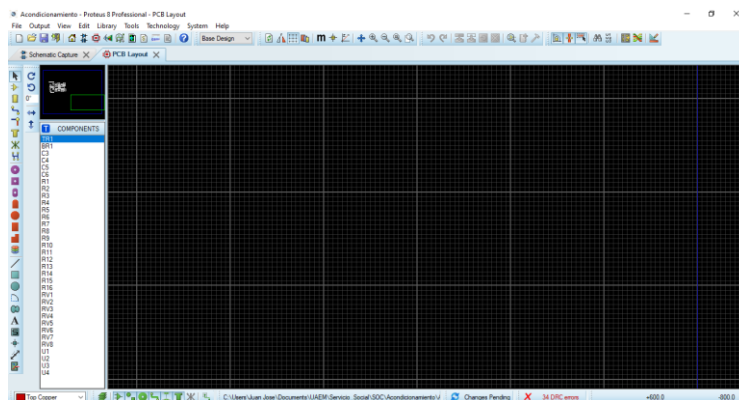
Hay que oprimir el botón sobre la P, e ir seleccionando uno por uno los componentes que son parte del circuito:



Los componentes se pegarán en el diseño esquemático y se alambrarán según el circuito deseado:

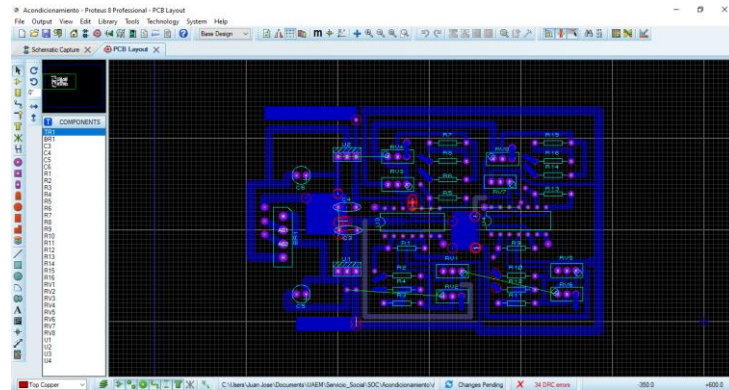


Para diseñar el PCB se tiene que ir a la vista del PCB Layout:

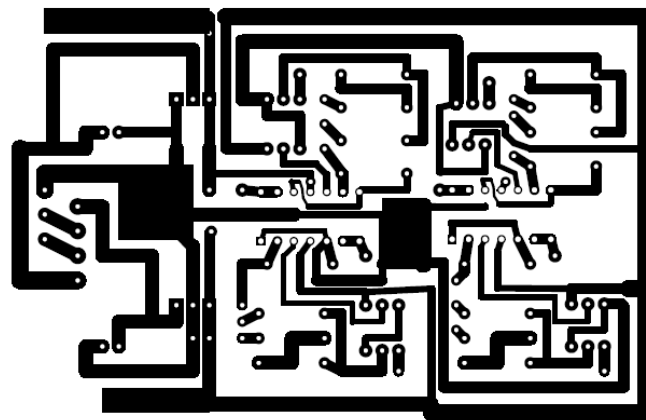


Ahora se deberá arrastrar y acomodar los componentes en el PCB Layout, el software indicará por medio de flechas que terminales van conectadas entre sí, según el diagrama esquemático lo indique. Para conectar hay dos opciones: unirlos manualmente o automáticamente, si se desea hacerlo manual solamente se van trazando las pistas del PCB con el mouse conectando un punto con otro, para hacerlo

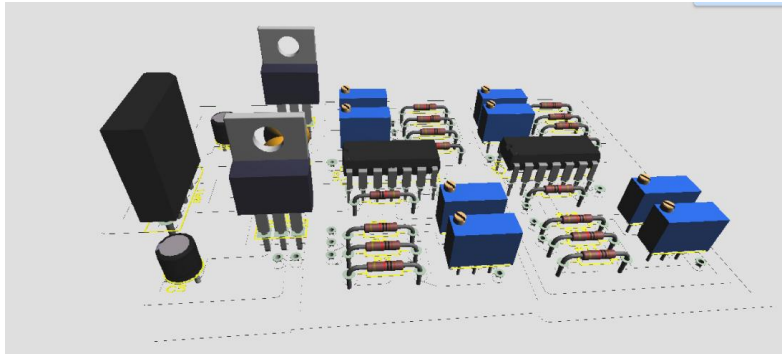
automáticamente Proteus cuenta con una opción llamada Auto Router, el cual conecta automáticamente los puntos que deben de ser conectados:



Para pasarlo a una placa fenólica el proceso que se realizó fue mandar el PCB a un PDF, imprimirlo en una hoja tipo stampa para que la tinta no se impregne en la hoja y se pueda pasar a la placa, luego aplicar el ácido para que se vote todo lo que no esté con la tinta, dejando así marcado el circuito en la placa:



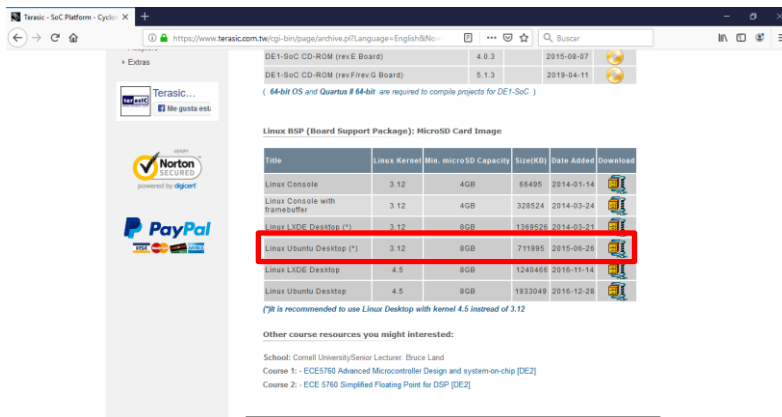
Finalmente, se taladró la placa y se soldaron bases para los circuitos integrados y los demás componentes dejando el circuito funcional:



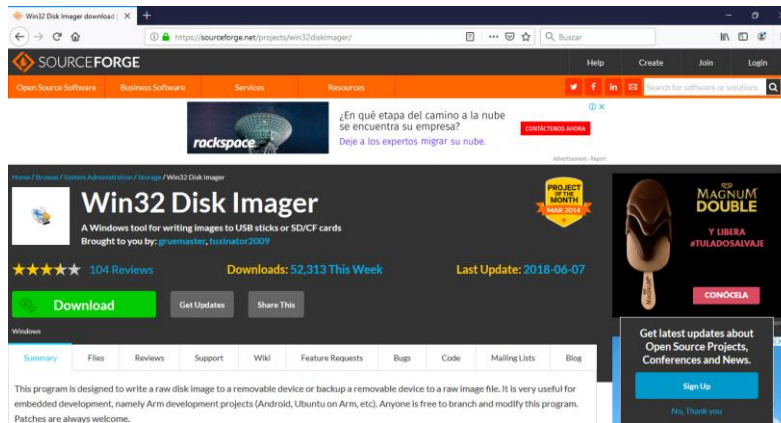
Una vez revisado el circuito se procedió a meterlo junto con los transformadores, el FPGA y conectores a un gabinete de modo que el sistema quedará sellado en un gabinete como de CPU.

C. MONTAR LINUX UBUNTU EN MICROSD

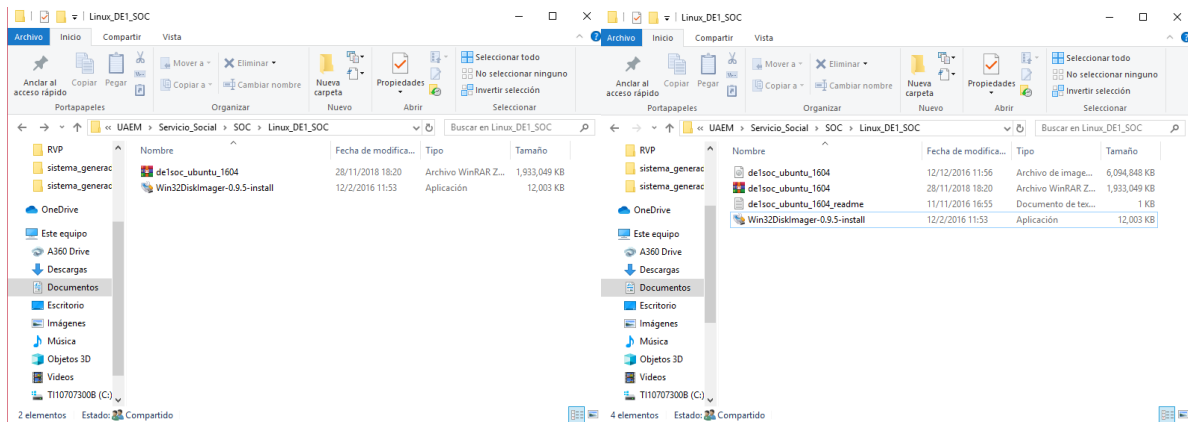
Lo primero que se tiene que hacer es descargar la imagen que se montará en la Micro SD [40]:



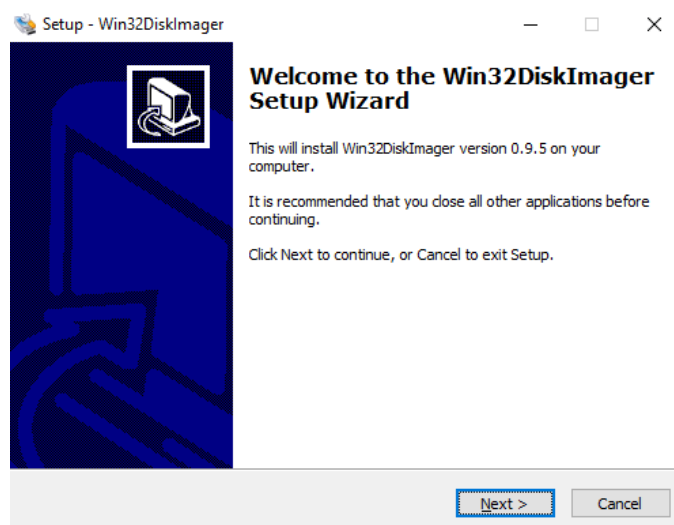
Luego se descargará la aplicación que montará la imagen en la micro SD, la aplicación se llama Win32DiskImager [41]:



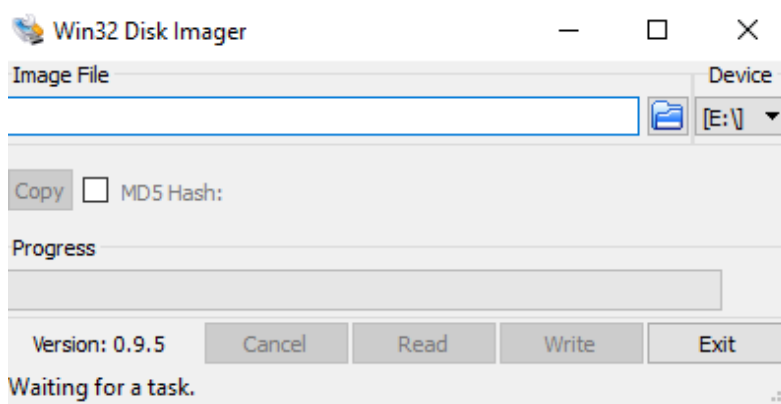
Una vez descargadas ambas cosas, se debe descomprimir el archivo de la imagen:



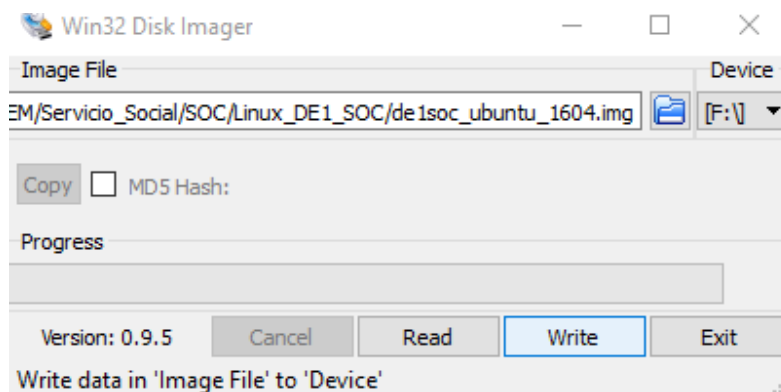
Luego se debe instalar la aplicación:



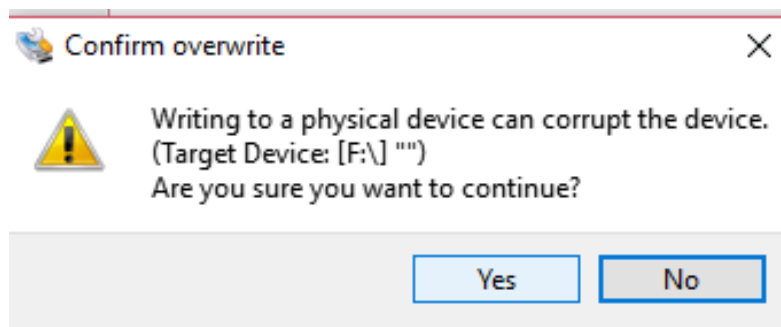
Una vez terminada la instalación, se debe abrir la aplicación:



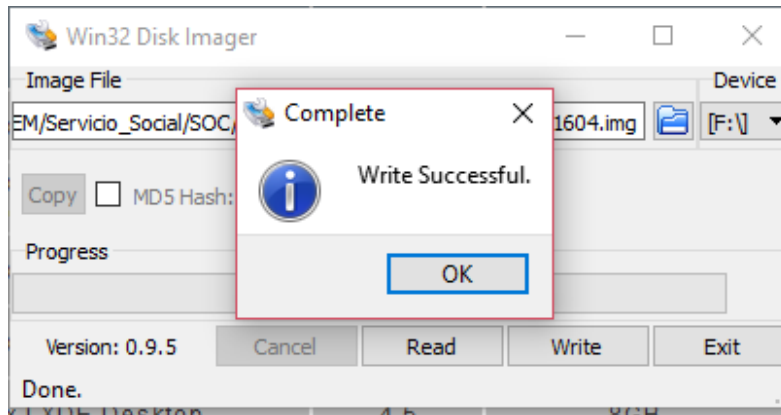
A continuación se selecciona en donde se montará, en este caso la microSD está en [F:\], y se busca el archivo de la imagen:



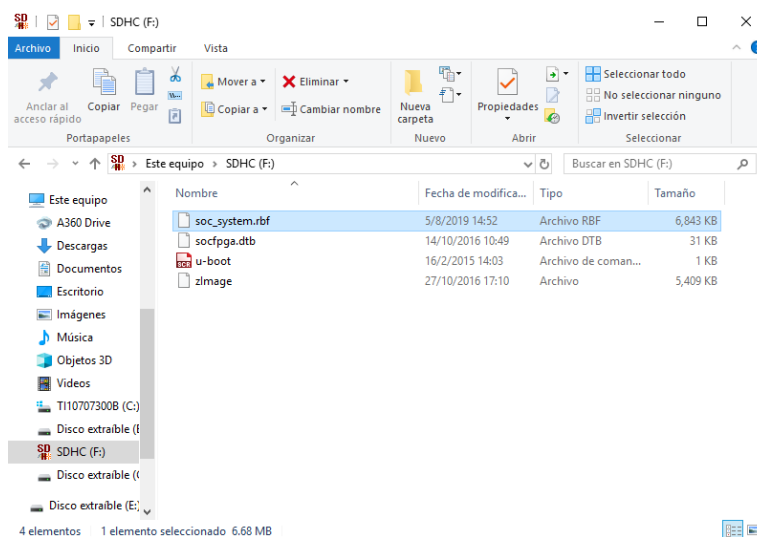
Se oprime el botón Write lo cual mostrará una advertencia que dice que lo que este en la microSD será formateado:



Cuando termine mostrará una ventana de que la imagen fue montada correctamente:

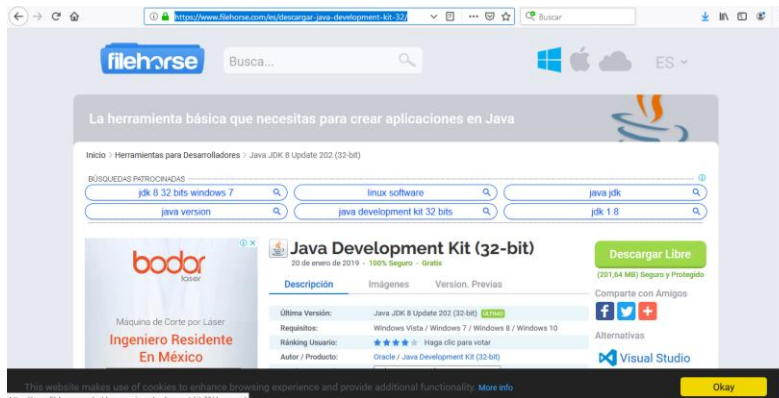


Si la microSD es leída se notará que tiene varios archivos instalados lo que da a entender que el montaje fue realizado con éxito:



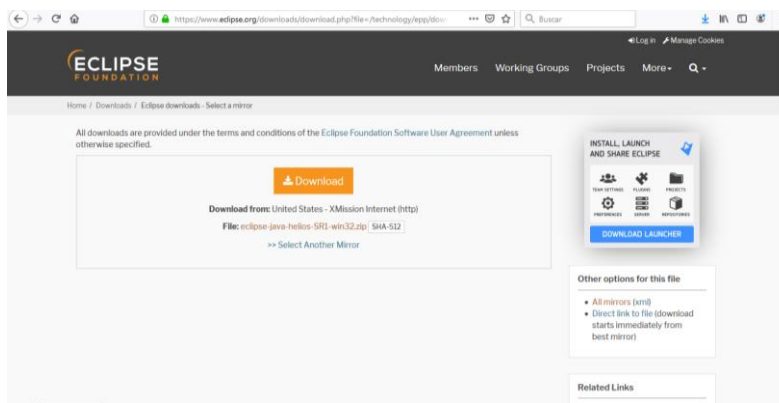
D. INSTALACIÓN DE ECLIPSE

Lo primero que se debe hacer es instalar el JDK. [42] Para descargarlo solo hay que oprimir el botón descargar libre. JDK significa Java Development Kit y es un conjunto de herramientas que te permite entre muchas otras cosas compilar programas en lenguaje Java:



De ahí se descargará el instalador, al cual solo se tendrá que dar Next a todo.

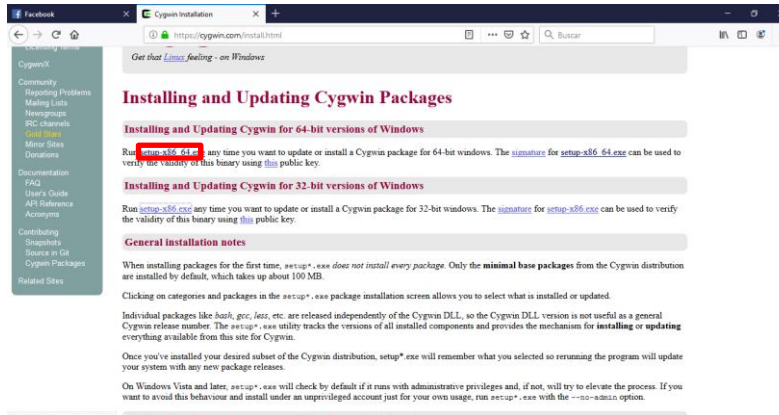
Después se debe descargar eclipse [43]:



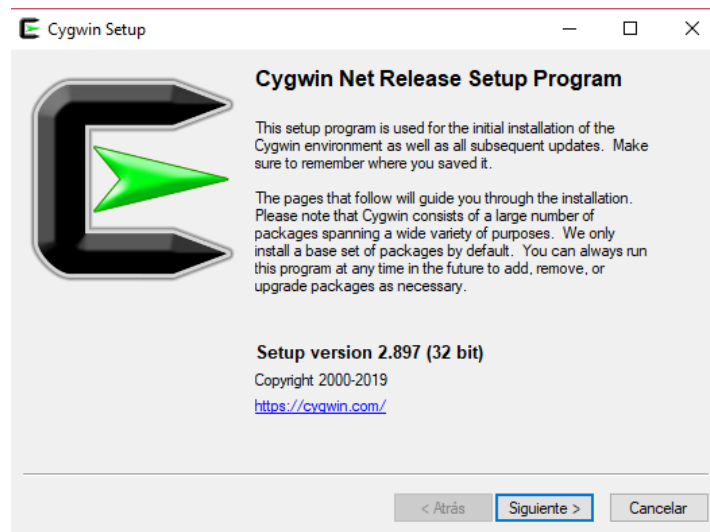
Se deberá oprimir el botón Download, el cual descargará un archivo .rar el cual solo bastará con descomprimirlo para que se instale Eclipse. Nota: Se tiene que instalar primero el JDK para poder abrir Eclipse. [44]

E. INSTALACIÓN DE CYGWIN

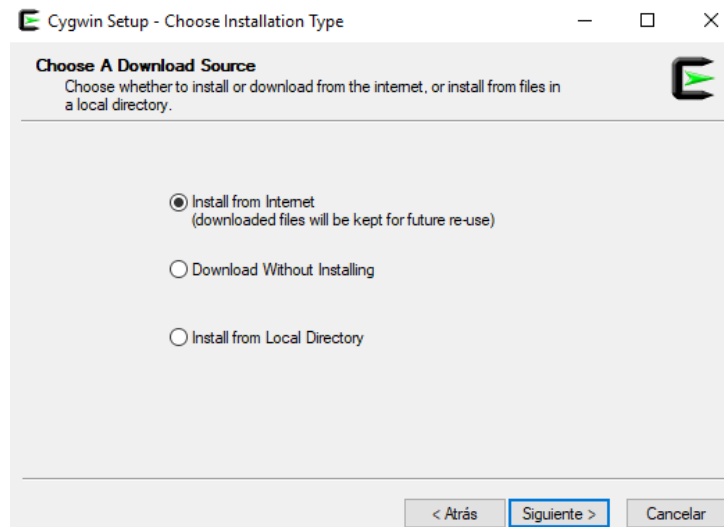
Cygwin se puede descargar de su página de internet [45]. Cygwin es un emulador de comandos de Linux que servirá para usar comandos de compilación de C:



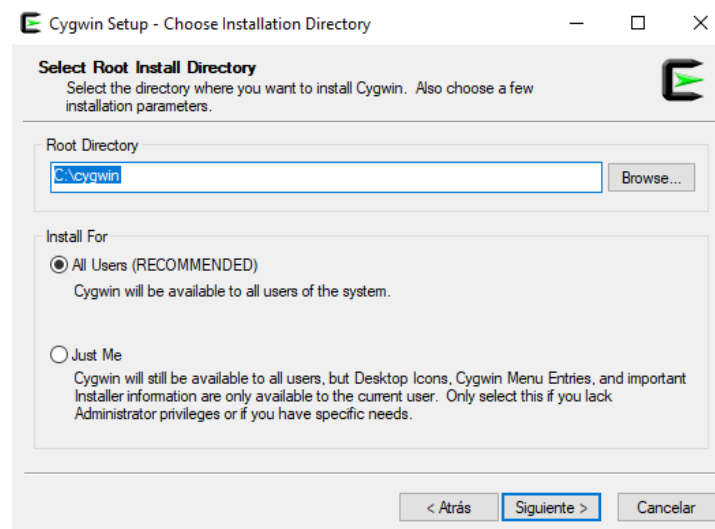
Se descargará el instalador y al abrirlo se abrirá la siguiente ventana y se oprimirá el botón siguiente.



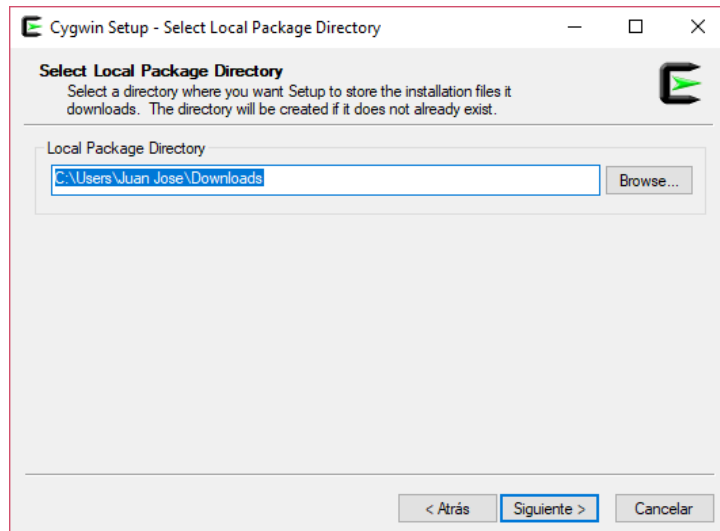
Se seleccionará la opción "Install from Internet" y se oprimirá el botón siguiente:



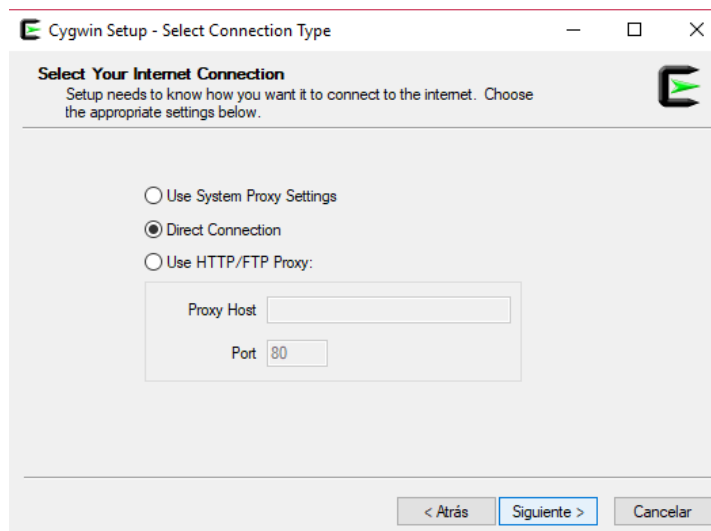
Luego se selecciona el directorio donde se guardará, se recomienda dejar el que marca por default:



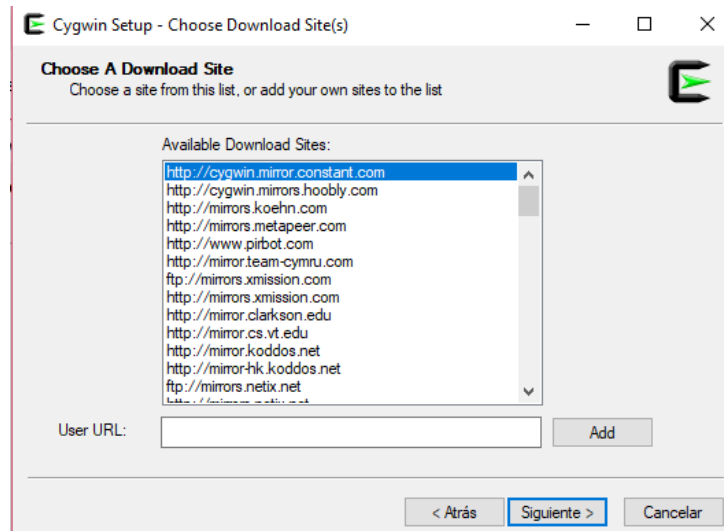
Luego se debe seleccionar una carpeta donde se guardaran los directorios para la instalación:



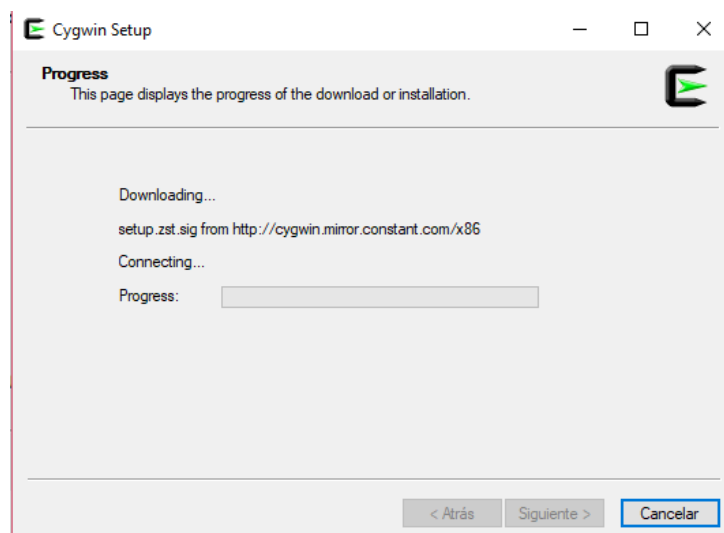
Luego se seleccionará la opción "Direct Conection":



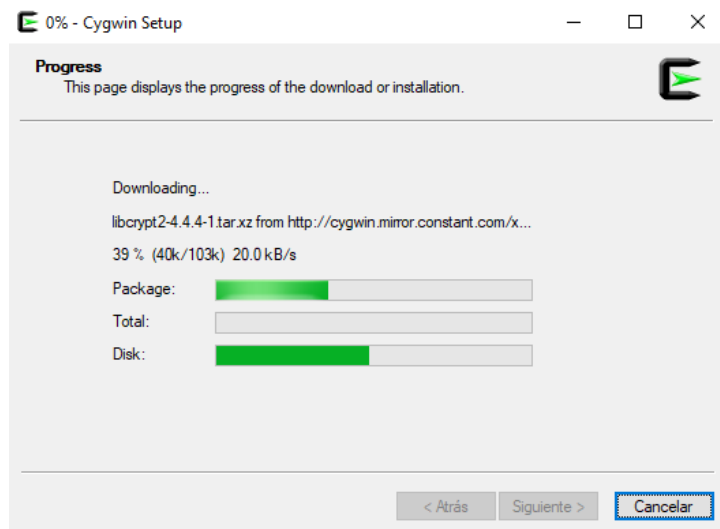
Se seleccionará cualquier página de la lista que aparecerá y se oprimirá el botón siguiente:



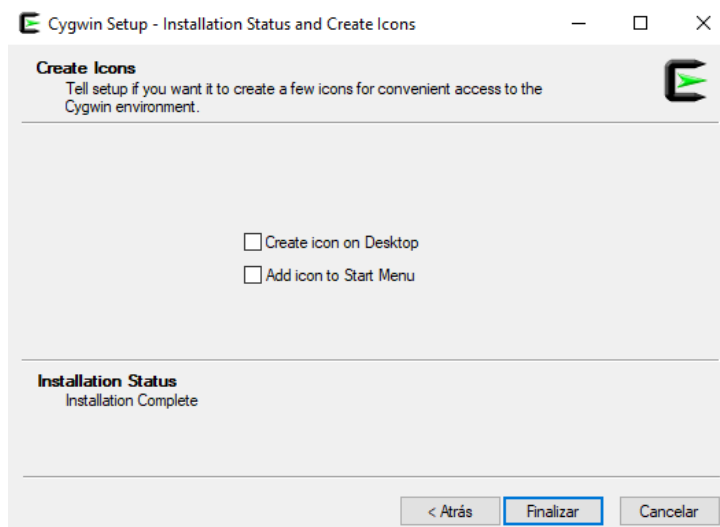
Los paquetes se empezarán a descargar:



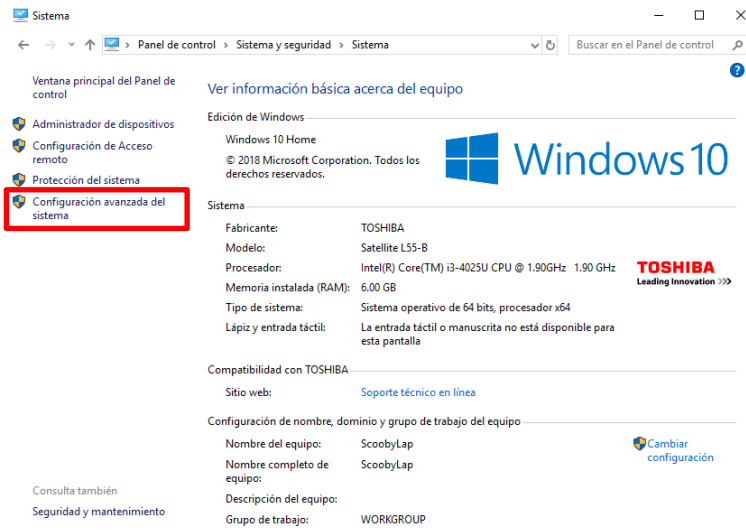
Cuando termine se abrirá una ventana en la cual se pueden seleccionar los paquetes que se desean descargar:



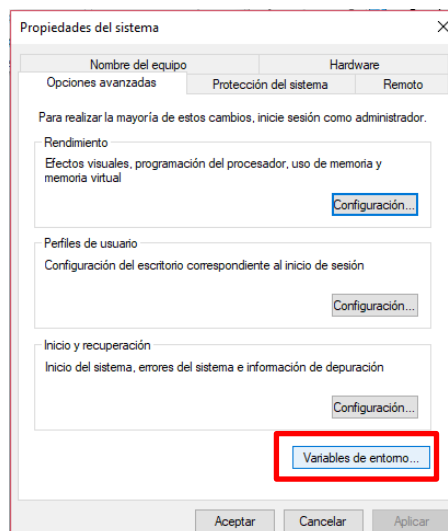
Cuando termine simplemente se oprimirá el botón Finalizar:



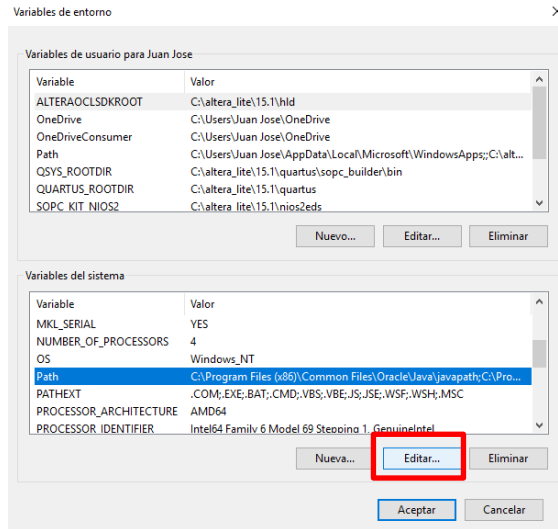
El siguiente paso será poner el path del Cygwin en las variables de entorno, para esto se debe ir al Panel de Control >> Sistema y Seguridad >> Sistema y ahí oprimir el botón Configuración avanzada del sistema:



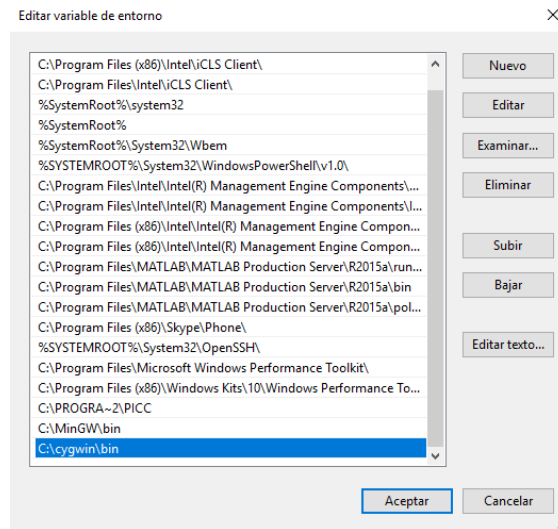
Se abrirá una ventana y se deberá oprimir el botón variables de entorno:



Se abrirá otra ventana en donde en el cuadro que tiene como título "Variables del sistema" se debe seleccionar la que se llama "Path" y oprimir el botón editar:

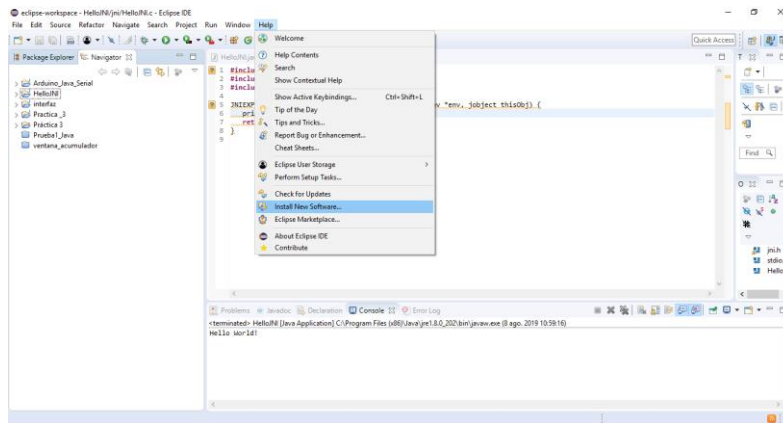


Se deberá oprimir el botón “Nuevo” y se escribirá la siguiente dirección:
“C:\cygwin\bin”:

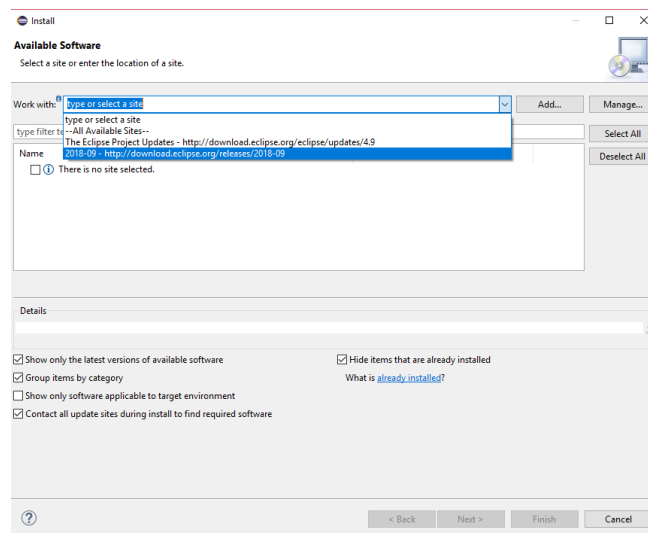


F. INSTALACIÓN DE CDT (C/C++ DEVELOPMENT TOOL)

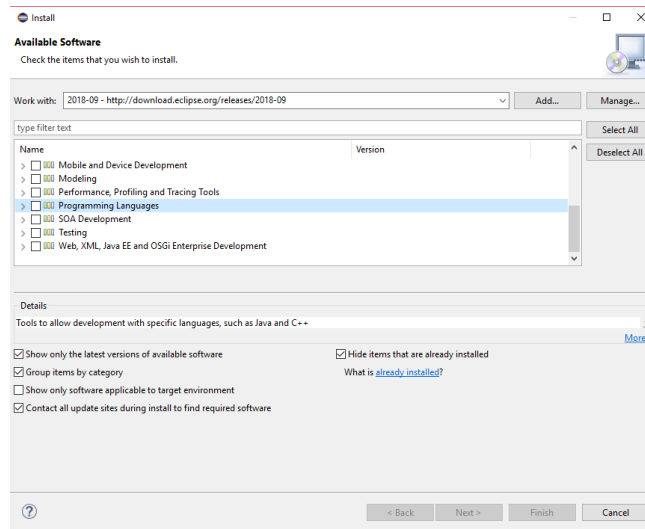
Dentro del eclipse se deberá oprimir el botón Help >> Install New Software [46]:



En la parte que dice Work with se debe seleccionar la opción que tenga el nombre de la versión de eclipse con la que se esté trabajando, en este caso es la 2018-09. Nota: Para este paso se requiere una conexión a Internet:

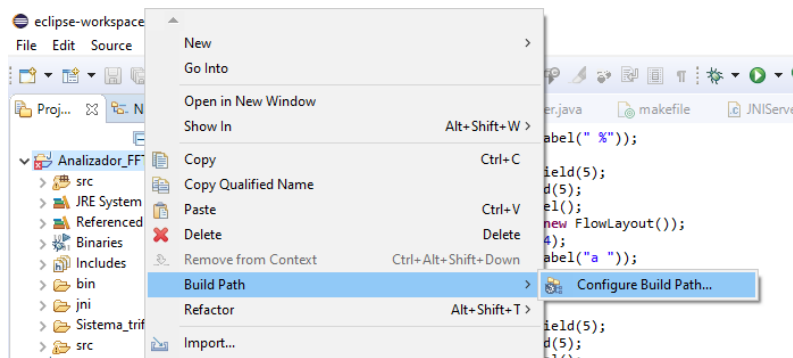


Luego se cargaran los paquetes que se pueden instalar en eclipse, se tiene que abrir el paquete que diga “Programming Languages” y seleccionar la opción que se llama “C/C++ Development Tools” oprimir el botón Next, aceptar las condiciones y esperar a que se instale el CDT:

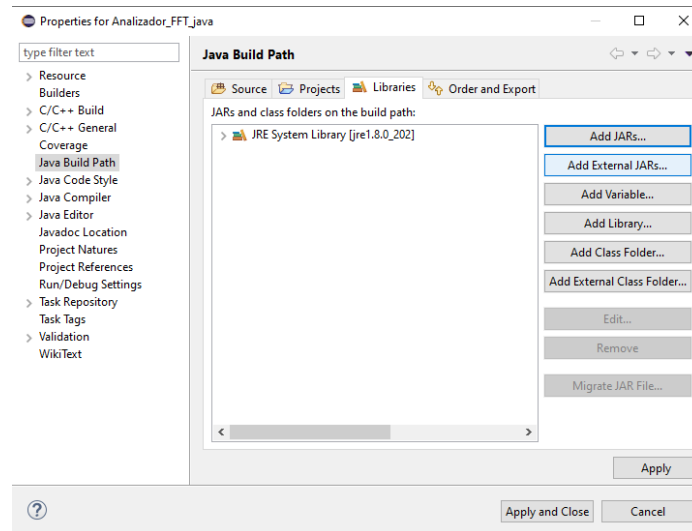


G. AGREGAR LIBRERÍAS A ECLIPSE

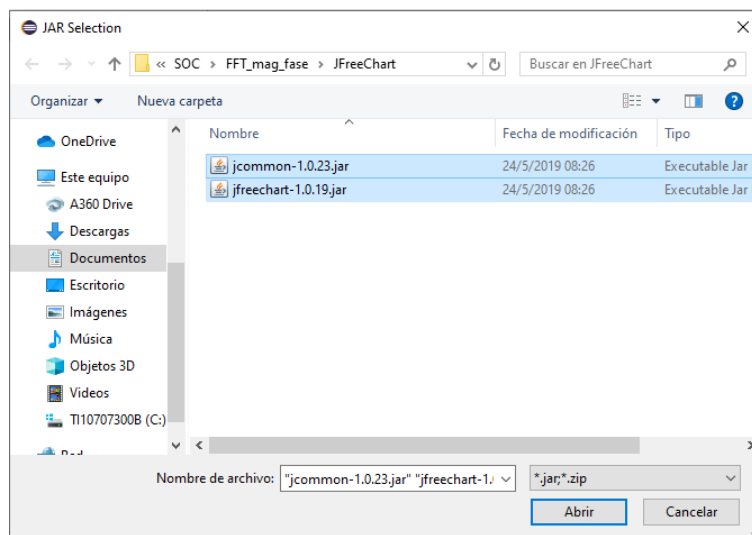
Para añadir una librería no incluida en eclipse a un proyecto, hay que oprimir el botón derecho del ratón en el nombre del proyecto, buscar la opción “Build Path” y luego seleccionar la opción “Configure Build Path”:



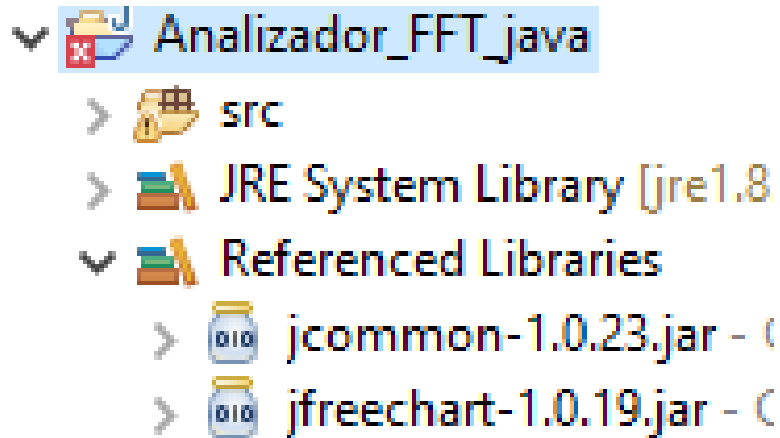
Se abrirá una ventana y se debe seleccionar la opción “Add External JARs”:



Ahora se deben seleccionar las librerías descargadas, las cuales deben estar en un formato .jar:

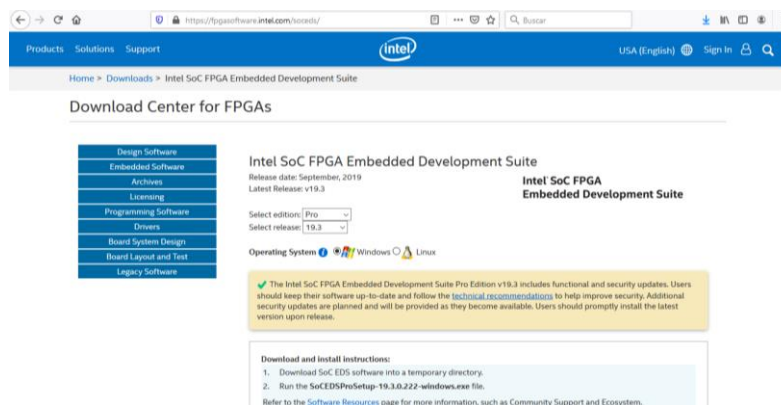


Luego de vuelta a la primera ventana solo hay que oprimir el botón aplicar. Para verificar que las librerías hayan sido incluidas de manera exitosa se deberá revisar que se haya creado una nueva carpeta en el proyecto que diga "Referenced Libraries" y ahí deberán estar las librerías añadidas:



H. DESCARGA DE SOC EDS (EMBEDDED DEVELOPMENT SUITE)

Para descargar se puede hacer directamente de la página de Intel [47]. La versión debe ser acorde a la versión de Quartus instalada:



I. INSTALACIÓN DE JAVA Y GCC EN LINUX UBUNTU

Primero para instalar java en Linux y también gcc, se requiere una conexión a internet. Para verificar que se encuentra el FPGA SoC conectado a Internet se debe usar el comando (escrito desde la terminal de Linux):

```
ifconfig
```

Y revisar si se tiene una dirección IP.

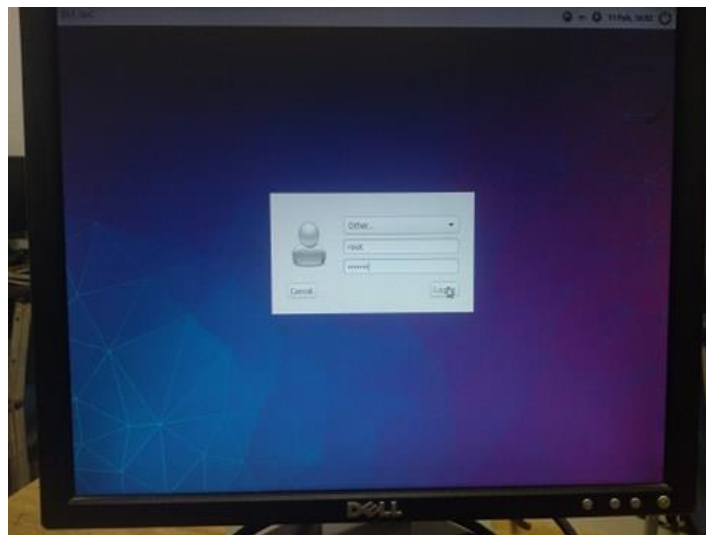
```
root@DE1_SoC: ~
File Edit Tabs Help
collisions:0 txqueuelen:1
RX bytes:5435 (5.4 KB) TX bytes:5435 (5.4 KB)

root@DE1_SoC:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 2a:d7:1f:00:f6:f2
          inet addr:148.218.46.112 Bcast:148.218.46.255 Mask:255.255.255.0
          inet6 addr: fe80::b3fc:3125:71d6:4d53/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:367 errors:0 dropped:0 overruns:0 frame:0
          TX packets:290 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:37782 (37.7 KB) TX bytes:27648 (27.6 KB)
          Interrupt:27 Base address:0xc000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:377 errors:0 dropped:0 overruns:0 frame:0
          TX packets:377 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:23779 (23.7 KB) TX bytes:23779 (23.7 KB)

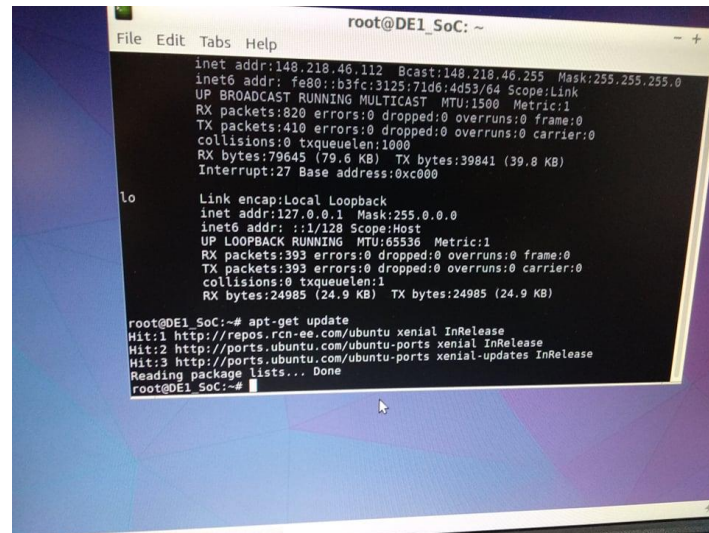
root@DE1_SoC:~#
```

Una vez que ya se tiene una conexión a Internet lo primero que se debe hacer es iniciar sesión con el root



Desde el root se debe abrir una terminal y escribir el comando:

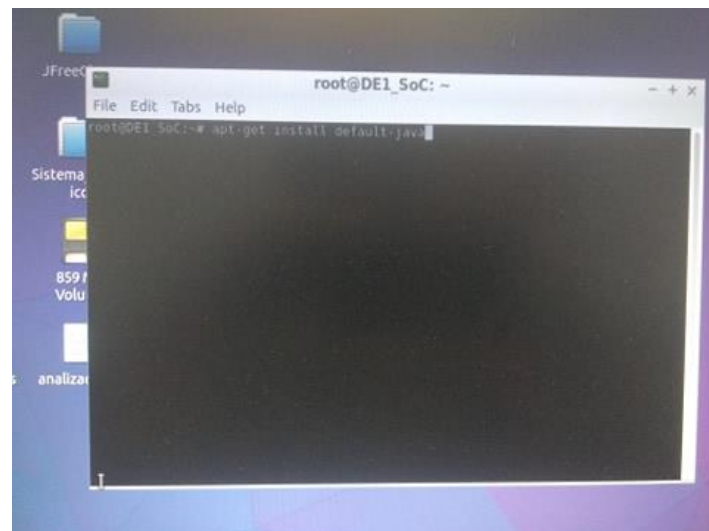
```
apt-get update
```



```
root@DE1_SoC: ~  
File Edit Tabs Help  
inet addr:148.218.46.112 Bcast:148.218.46.255 Mask:255.255.255.0  
inet6 addr: fe80::b3fc:3125:71d6:4d53/64 Scope:Link  
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
RX packets:820 errors:0 dropped:0 overruns:0 carrier:0  
TX packets:410 errors:0 dropped:0 overruns:0 carrier:0  
collisions:0 txqueuelen:1000  
RX bytes:79645 (79.6 KB) TX bytes:39841 (39.8 KB)  
Interrupt:27 Base address:0xc000  
  
lo  
Link encap:Local Loopback  
inet addr:127.0.0.1 Mask:255.0.0.0  
inet6 addr: ::1/128 Scope:Host  
UP LOOPBACK RUNNING MTU:65536 Metric:1  
RX packets:393 errors:0 dropped:0 overruns:0 frame:0  
TX packets:393 errors:0 dropped:0 overruns:0 carrier:0  
collisions:0 txqueuelen:1  
RX bytes:24985 (24.9 KB) TX bytes:24985 (24.9 KB)  
  
root@DE1_SoC:~# apt-get update  
Hit:1 http://repos.rcn-ee.com/ubuntu xenial InRelease  
Hit:2 http://ports.ubuntu.com/ubuntu-ports xenial InRelease  
Hit:3 http://ports.ubuntu.com/ubuntu-ports xenial-updates InRelease  
Reading package lists... Done  
root@DE1_SoC:~#
```

Ahora se debe escribir el siguiente comando para instalar Java [48]:

```
apt-get install default-java
```



```
root@DE1_SoC:~# apt-get install default-java
```

Después empezará a buscar el java en internet, y cuando lo encuentre preguntará si se desea continuar a lo que se le deberá poner "y". Se pondrá a descargar e instalar el Java y cuando termine se puede comprobar su correcta instalación con el comando:

```
java -version
```



```
ubuntu@DE1_SoC: ~  
File Edit Tabs Help  
ubuntu@DE1_SoC:~$ java -version  
openjdk version "1.8.0_222"  
OpenJDK Runtime Environment (build 1.8.0_222-8u222-b10-1ubuntu1-16.04.1-b10)  
OpenJDK Client VM (build 25.222-b10, mixed mode)  
ubuntu@DE1_SoC:~$
```

Para instalar gcc se debe escribir el siguiente comando: [49]

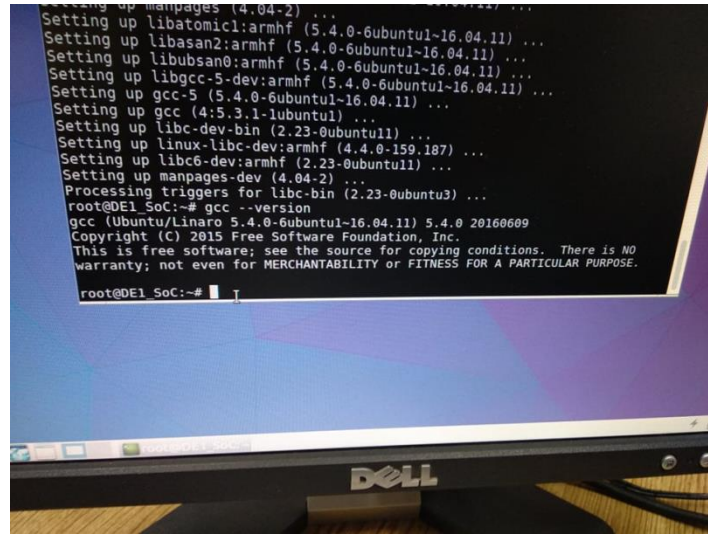
```
apt install gcc
```

```
root@DE1_SoC: ~  
File Edit Tabs Help  
Hit:3 http://ports.ubuntu.com/ubuntu-ports xenial-updates InRelease  
Reading package lists... Done  
root@DE1_SoC:~# apt-get install gcc  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following additional packages will be installed:  
  cpp-5 gcc-5 gcc-5-base libasan2 libatomic1 libc-dev-bin libc6 libc6-dbg  
  libc6-dev libcc1-0 libgcc-5-dev libgomp1 libstdc++6 libubsan0 linux-libc-dev  
  manpages manpages-dev  
Suggested packages:  
  gcc-5-locales gcc-multilib autoconf automake libtool flex bison gcc-doc  
  gcc-5-multilib gcc-5-doc libgcc1-dbg libgomp1-dbg libitm1-dbg libatomic1-dbg  
  libasan2-dbg liblsan0-dbg libtsan0-dbg libubsan0-dbg libcilkrts5-dbg  
  libmpx0-dbg libquadmath0-dbg glibc-doc  
The following NEW packages will be installed:  
  gcc gcc-5 libasan2 libatomic1 libc-dev-bin libc6-dev libgcc-5-dev libubsan0  
  linux-libc-dev manpages manpages-dev  
The following packages will be upgraded:  
  cpp-5 gcc-5-base libc6 libc6-dbg libcc1-0 libgomp1 libstdc++6  
7 upgraded, 11 newly installed, 0 to remove and 573 not upgraded.  
Need to get 0 B/25.4 MB of archives.  
After this operation, 37.9 MB of additional disk space will be used.  
Do you want to continue? [Y/n]
```

Después empezará a buscar el gcc en internet, y cuando lo encuentre preguntará si se desea continuar a lo que se le deberá poner "y"

Se pondrá a descargar e instalar el gcc y cuando termine se puede comprobar su correcta instalación con el comando

```
gcc --version
```



BIBLIOGRAFÍA

- [1] Douglas J. Smith, "HDL Chip Design", Doone Publications, pp.8, 1996.
- [2] <https://www.verilog.com/>
- [3] Altera Corporation, "Quartus II Handbook Volume 1: Design and Synthesis", 2015.
- [4] Huircán Juan Ignacio, "Conversores Análogo-Digital y Digital-Análogo: Conceptos Básicos", Departamento de Ingeniería Eléctrica Universidad de la Frontera, pp.1-7, 2007.
- [5] Terasic Technologies Inc., "DE1 SoC User Manual", Altera University Program, 2014.
- [6] Linear Technology Corporation, "LTC2308", 2007.
- [7] Romero Troncoso, "Electrónica Digital y Lógica Programable", Universidad de Guanajuato (UGto), pp.520-528, 2016.
- [8] S. A. Tretter, Communication System Design Using DSP Algorithms, New York, USA: Kluwer Academic / Plenum Publishers, 2003, pp. 109-120.
- [9] Thomas L. Floyd, "Fundamentos de Sistemas Digitales", Prentice Hall, pp. 644-646, 2006.
- [10] <https://es.mathworks.com/>
- [11] http://www.eudim.uta.cl/files/8313/2215/7786/fm_Ch04_mfuentesm.pdf
- [12] Hyat, W. H., Kemmerly, J. E., & Durbin, S. M. (2007). Análisis de circuitos para ingeniería 7ed. Ciudad de México: McGrawHill.
- [13] <https://www.labcenter.com/>
- [14] Bob Zeidman, "Designing with FPGAs and CPLDs", CMP Books, pp.17-51, 2002.
- [15] <https://sirinsoftware.com/blog/the-arm-processor-a-r-and-m-categories-and-their-specifcs/>

- [16] David J Greaves, "System on Chip Design and Modelling", University of Cambridge, pp.1-3, 2008.
- [17] Altera Corporation, "What is a SoC FPGA", Architecture Brief, 2014
- [18] Altera Corporation, "Cyclone V Hard Processor System Technical Reference Manual", Intel, pp. 40, 610-665, 2018
- [19] Patricio Galeas, "Linux para Sistemas Embebidos", GitHub, 2018
- [20] https://github.com/thinkoco/de1soc_media
- [21] Altera Corporation - University Program. (2014). Making Qsys Components.
- [22] Altera Corporation - University Program. (2014). DE1-SoC Getting Started Guide.
- [23] Enrique Vicente Bonet Esteban, "Lenguaje C", Universitat de València, pp.1-2, 2010
- [24] Altera Corp. "Configuring HPS to FPGA and FPGA to HPS Bridges in Altera SoCs.
- [25] Carlos González Morcillo, Miguel Ángel Redondo Duque. "Punteros en C", Escuela Superior de Informática - Ciudad Real Universidad de Castilla-La Mancha, pp. 1-8, 2003.
- [26] Depto. Ciencias de la Computación e IA, "Introducción al lenguaje Java", Universidad de Alicante, 2013.
- [27] Pilar Aranzazu Ocaña Díaz-Ufano, José Miguel Ordaz Cassá, "Programación en Java avanzado", Instituto Nacional de Tecnologías Educativas y de Formación del Profesorado, pp. 5, 2011.
- [28] Brian Cole, Robert Eckstein, James Elliott, Marc Loy, David Wood, "Java™ Swing", O'Reilly, pp.1, 2002.
- [29] Sheng Liang, "The Java™ Native Interface Programmer's Guide and Specification", Addison Wesley Longman, pp. 3-6, 1999.

- [30] Java Code Geek, "Eclipse IDE Handbook", Exelixis Media P.C., pp.1, 2017
- [31] <https://communities.bmc.com/thread/102236>
- [32] <http://www.jfree.org/jfreechart/>
- [33] <https://stackoverflow.com/questions/34251173/calling-c-code-from-frame-in-java>
- [34] <https://stackoverflow.com/questions/3451378/how-to-run-javah-from-eclipse>
- [35] <https://forums.intel.com/s/question/0D50P00003yyT5zSAE/sopccreateheaderfiles-command-not-found-?language=es>
- [36] https://www.tutorialspoint.com/jfreechart/jfreechart_installation.htm
- [37] <https://www3.ntu.edu.sg/home/ehchua/programming/java/JavaNativeInterface.html>
- [38] <https://www.ni.com/es-mx/shop/electronic-test-instrumentation/application-software-for-electronic-test-and-instrumentation-category/what-is-multisim.html>
- [39] <https://www.intel.com/content/www/us/en/programmable/downloads/download-center.html>
- [40] <https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=836&PartNo=4>
- [41] <https://sourceforge.net/projects/win32diskimager/>
- [42] <https://www.filehorse.com/es/descargar-java-development-kit-32/>
- [43] <https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/helios/SR1/eclipse-java-helios-SR1-win32.zip>
- [44] <https://stackoverflow.com/questions/11461607/cant-start-eclipse-java-was-started-but-returned-exit-code-13>
- [45] <https://cygwin.com/install.html>
- [46] Eclipse's "C/C++ Development Tool User Guide", accessible via Eclipse's Help menu.

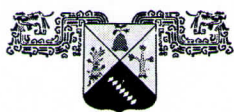
[47] <https://fpgasoftware.intel.com/soceds/>

[48] <https://www.digitalocean.com/community/tutorials/como-instalar-java-con-apt-get-en-ubuntu-16-04-es>

[49] <https://linuxconfig.org/how-to-install-gcc-the-c-compiler-on-ubuntu-18-04-bionicbeaver-linux>

[50] YHDC, "SCT-013 Datasheet".

[51] Comité Consultivo Nacional de Normalización de Instalaciones Eléctricas, Distribución y Abastecimiento de Energía Eléctrica, y Recursos Nucleares, "NORMA Oficial Mexicana NOM-001-SEDE-2012, Instalaciones Eléctricas", Diario Oficial de la Federación, pp. 138-747, 2012.



UNIVERSIDAD AUTÓNOMA DEL
ESTADO DE MORELOS

FACULTAD DE CIENCIAS QUÍMICAS e INGENIERÍA

Programas educativos de calidad reconocidos por CIEES, CACEI y CONACYT
SGC certificado en la norma ISO 9000:2015



Facultad de Ciencias
Químicas e Ingeniería

Cuernavaca, Morelos, 18 de Febrero de 2020

**FORMA T-4A
NOMBRAMIENTO COMITÉ REVISOR**

**DR. LUIS CISNEROS VILLALOBOS
DR. J GUADALUPE VELÁSQUEZ AGUILAR
DR. MARIO LIMÓN MENDOZA
DR. JOSÉ GERARDO VERA DIMAS
DR. J. JESÚS ESCOBEDO ALATORRE**

P R E S E N T E

Me permito comunicarles que han sido designados integrantes del **COMITÉ REVISOR** del trabajo de:

TESIS

Titulado:

ANALIZADOR MULTICANAL DE SEÑALES ELÉCTRICAS BASADO EN SOC FPGA

Que presenta (el) o (la) **C. LÓPEZ SOLORZANO JUAN JOSÉ**, del programa educativo de: **INGENIERÍA ELÉCTRICA-ELECTRÓNICA**

ATENTAMENTE
Por una humanidad culta

**DRA. VIRIDIANA A. LEÓN HERNÁNDEZ
DIRECTORA**

DICTAMEN

**DRA. VIRIDIANA A. LEÓN HERNÁNDEZ
DIRECTORA DE LA FCQeI
P R E S E N T E**

En respuesta a su amable solicitud para emitir DICTÁMEN sobre el trabajo que se menciona, me permito informarle que nuestro voto es:

VOTO	NOMBRE	FIRMA
Aprobado	DR. LUIS CISNEROS VILLALOBOS	
Aprobado	DR. MARIO LIMÓN MENDOZA	
Aprobado	DR. J GUADALUPE VELÁSQUEZ AGUILAR	
Aprobado	DR. J. JESÚS ESCOBEDO ALATORRE	
Aprobado	DR. JOSÉ GERARDO VERA DIMAS	

C.c.p. – Archivo.