ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"GENERADOR DE SEÑALES BÁSICAS, UTILIZANDO LA PLATAFORMA DE NIOS II"

INFORME DEL PROYECTO DE GRADUACIÓN

Previa la obtención del Título de:

INGENIERO EN TELEMÁTICA

Presentado por:

Roberto Andrés Noboa Gárate

Sergio David Chamba Aguas

GUAYAQUIL – ECUADOR

Año 2015

AGRADECIMIENTO

A Dios por todas las bendiciones que me ha dado.

A mis padres por el apoyo incondicional para realizar mis estudios, en especial a mi madre que me enseño que ante la adversidad jamás se debe bajar los brazos, a mis abuelitos por darme esa guía y fortaleza en todo momento.

A mi enamorada por darme su apoyo constante en esta etapa.

Al M.Sc. Ponguillo por su guía, enseñanzas, tolerancia y apoyo que ha sido de vital para culminar este proyecto con éxito.

Roberto Andrés Noboa Gárate

A mis abuelitos, siempre estarán en mi mente y son mi motivación para seguir adelante.

Sergio David Chamba Aguas

DEDICATORIAS

En primer lugar a Dios, a mi familia a mis seres queridos.

Roberto Andrés Noboa Gárate

A aquellas personas que en los momentos más difíciles estuvieron ahí, apoyándome.

Sergio David Chamba Aguas

TRIBUNAL DE SUSTENTACIÓN

Ph.D. Sixto García

SUBDECANO DE LA FIEC

M.Sc. Ronald Ponguillo

DIRECTOR DEL PROYECTO DE GRADUACIÓN

M.Sc. Víctor Asanza

MIEMBRO PRINCIPAL

DECLARACIÓN EXPRESA

“La responsabilidad del contenido de este Informe, nos corresponde exclusivamente; y el patrimonio intelectual del mismo a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL” (Reglamento de exámenes y títulos profesionales de la ESPOL).

_____

Roberto Andrés Noboa Gárate

_____

Sergio David Chamba Aguas

RESUMEN

Este proyecto contiene la implementación de un **GENERADOR DE SEÑALES DIGITALES BÁSICAS UTILIZANDO LA PLATAFORMA DE NIOS II** para fines académicos, el mismo que genera señales sinoidales, triangulares, cuadradas y rampa, usando la técnica del Sintetizador Digital Directo DDS (Síntesis Digital Directa), o también conocido como NCO (Oscilador Numéricamente controlado).Para el desarrollo del proyecto hemos utilizado la tarjeta de desarrollo DE-0 Nano de Altera basada en un dispositivo Cyclone IV® EP4C22 FPGA, en el que se ha diseñado la arquitectura de un mini-computador basado en el Microprocesador NIOS II.

En el desarrollo del proyecto cabe resaltar dos partes fundamentales: La primera se basa en el diseño del hardware donde se agrega el CORE "WAVEFORM_GEN/NCO" de OpenCores que es un acumulador de fase donde carga en cada incremento una tabla LUT ("Look-up Table") para generar las señales. La segunda parte corresponde a la implementación del software en base al diseño del hardware. Mediante la programación de la aplicación embebida, presentamos en un LCD-Touch una interfaz gráfica para que el usuario pueda seleccionar los parámetros de la señal a generar.

ÍNDICE GENERAL

ABREVIATURAS Y SIMBOLOGÍA

DAC	Convertidor Digital – Analógico
DE0-NANO	Tarjeta de Desarrollo
DDS	Síntesis Digital Directa
FPGA	Arreglo de puertas lógicas programables
GPIO	Puerto de propósito general Input/Output
GUI	Interfaz Gráfica de Usuario
HDL	Lenguaje de descripción de hardware
JTAG	Joint Test Action Group
LCD	LiquidCrystalDisplay
LUT	Tabla de consulta (LookupTable)
NCO	Oscilador Controlado Numéricamente
PCB	Tarjeta de Circuito Impresa
QSYS	QuadSystemsCorporation
SOPC	Sistema en Chip Programable

SOC	Sistema en Chip
SOTR	Sistema operativo de tiempo real
TTL	Lógica Transistor Transistor
UART	Universal Asynchronous Receiver

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Con el avance de la tecnología los sistemas embebidos van consolidándose protagonizando el área de desarrollo de dispositivos electrónicos, ya que nos brindan muchas herramientas importantes, dentro de las mismas cabe destacar al procesador de NIOS II el cual nos presenta múltiples funcionalidades especiales para el desarrollo de hardware.

Las FPGAs (Field Programable Gate Array) son utilizadas en los sistemas embebidos ya que son una alternativa a los microcontroladores por su flexibilidad, la posibilidad de reprogramación del hardware y software, la reducción de costos que puede representar e incorporar en la misma (FPGA) funcionalidades que antes eran realizadas por distintos integrados que hacían a la tarjeta de circuito impreso más compleja; esto hace que hoy en día se pueda evaluar seriamente el uso de las FPGA en estos sistemas.

El proyecto consta de la implementación de un Generador de Señales, que es muy útil en los laboratorios de electrónica analógica, digital y en el campo de las telecomunicaciones, este equipo es una herramienta

vital para las diferentes pruebas y experimentos de instrumentación ya que es el mejor para realizar simulaciones con señales.

El proyecto se basa en la técnica DDS, la mismo que genera valores digitales de onda a partir de una frecuencia de reloj fija, de esta manera genera señales de frecuencia ajustable, las características principales de un DDS son: un registro de fase, un acumulador de fase y una tabla (LUT), que trabajan conjuntamente con un DAC (Convertidor Digital Análogo) para obtener finalmente la señal análoga deseada.

Se lo ha estructurado en 4 capítulos que los detallamos a continuación:

En el capítulo 1: Se presenta el planteamiento del problema, los objetivos generales, objetivos específicos, los alcances, limitaciones del proyecto y la metodología empleada.

En el capítulo 2: Contiene el Marco Teórico en el que se basa el proyecto. Se describen los conceptos esenciales de los SoC (Sistemas en Chips Configurables), los sistemas embebidos, el procesador de

NIOS II, el CORE WAVEFORM_GEN/NCO, Módulo LCD-Touch, FPGAs, DDS, DAC, uC/OS-II.

En el capítulo 3: Se detalla el Diseño e Implementación del proyecto, describe cada etapa y también los componentes que se utilizaron para la elaboración del sistema.

En el capítulo 4: Se presentan las pruebas y el análisis de los resultados las ventajas y desventajas del proyecto.

Para finalizar se muestran las conclusiones obtenidas en el desarrollo del proyectos y sus respectivas recomendaciones.

CAPÍTULO 1

1. GENERALIDADES

En este capítulo se detalla el planteamiento del problema, requerimientos y se describe cuáles son sus objetivos; así como también se presenta los alcances y limitaciones del proyecto.

1.1 IDENTIFICACIÓN DEL PROBLEMA

Dada la necesidad de desarrollar un equipo generador de señales portable, ya que es de nuestro conocimiento que el laboratorio de digitales no cuenta con uno, que nos permita realizar pruebas, experimentos de instrumentación, prácticas de laboratorio con señales de prueba y apoyo didáctico en la realización de proyectos.

1.2 ALCANCES Y LIMITACIONES

Los alcances de este proyecto son:

- La selección de los parámetros (frecuencia y amplitud), del generador de señales se la realizará mediante la interfaz gráfica mostrada en la pantalla LCD-Touch.
- Implementación del CORE WAVEFORM_GEN/NCO del portal web OpenCores para generar las señales.

Las limitaciones de este proyecto son las siguientes:

- El rango de frecuencias es: 1 [Hz] - 500 [kHz].
- Configuración del DAC (AD767) y velocidad de conversión que en sus especificaciones detalla un máximo de 4us.

- El rango de amplitud es de 0 – 10 [Vpp].

1.3 JUSTIFICACIONES

Los sistemas embebidos han experimentado un gran desarrollo dentro de la electrónica. El desarrollo de sistemas embebidos modernos demanda la implementación de funciones sofisticadas en plazos de diseño cortos. Estos sistemas poseen generalmente uno o más microprocesadores, memorias y periféricos cumplen con tareas específicas y pueden formar parte de sistemas complejos.

Dentro de las prioridades de instrumentos electrónicos en un laboratorio de electrónica está el **GENERADOR DE SEÑALES**. Los sistemas embebidos son una gran herramienta para la construcción de dispositivos electrónicos complejos, por eso implementaremos un generador de señales digitales para fines académicos, con pantalla táctil e interfaz amigable al usuario utilizando esta tecnología

Una parte muy importante en la elaboración del proyecto es encontrar un sentido no sólo académico, son también que este sea útil para

futuras generaciones con el fin de fomentar la investigación de los sistemas embebidos y desarrollo de sistemas más complejos.

1.4 SOLUCIÓN PROPUESTA

Nuestra solución al proyecto se enfocará en dos fases principales para obtener el producto deseado:

- La primera fase: Generar señales mediante hardware utilizando los recursos de la FPGA y el procesador NIOS II. En esta fase se elige el procesador NIOS II a utilizar con todas las respectivas configuraciones en base a nuestro requerimiento; luego se integra el CORE WAVEFORM_GEN/NCO que se basa en la tecnología DDS el cual fue personalizado para nuestras necesidades de diseño de hardware y adquisición de datos según la configuración e interpretación de datos utilizada en el DAC (AD767).

- En la segunda fase: Consiste en la programación del software utilizando el IDE NIOS II SBT, para que realice las funcionalidades en base al diseño de hardware. El software utiliza un sistema operativo en tiempo real (RTOS) llamado **“uC/OS II”** que permite realizar múltiples tareas a la vez (Multi-

tasking). Se diseño una interfaz gráfica (GUI) en la pantalla táctil la cual muestra un menú para la selección de los parámetros de las señales a generar.

1.5 OBJETIVOS

1.5.1 OBJETIVOS GENERALES

- Diseñar e implementar un sistema embebido basado en el procesador NIOS II y bloques de lógica digital configurable que funcione como un generador de señales básicas para fines académicos de laboratorio.

- Integrar al sistema de NIOS II, el CORE **WAVEFORM_GEN/NCO** que aplica la técnica de un DDS, siendo este método el más eficaz para obtener señales precisas y de menor rango de distorsión.

- Determinar las ventajas y desventajas que nos brinda nuestro generador de señales.

1.5.2 OBJETIVOS ESPECÍFICOS

- Construir un dispositivo que cumpla con todas las funciones de un generador de señales básico.

- Diseñar el sistema basado en el procesador de NIOS II.
- Reforzar los conocimientos sobre el funcionamiento de la FPGA, dominar la lógica programable, lenguajes de descripción de hardware para resolver problemas de manera óptima.
- Utilizar los recursos de la tarjeta de Desarrollo DE0-NANO para la implementación del proyecto.
- Integrar el CORE WAVEFORM_GEN/NCO al hardware del sistema.
- Crear el algoritmo y el código en C para el funcionamiento del sistema.
- Aprender el funcionamiento de los convertidores DAC (Convertidor Digital Análogo).
- Dominar la aplicación de multi-task, que nos brinda el sistema operativo en tiempo real **"uC/OS-II".**

1.6 METODOLOGÍA

En esta sección se analizarán las técnicas empleadas para el desarrollo del equipo propuesto que consta en las siguientes etapas:

- Etapa 1: Análisis de equipos generadores de señales existentes para plantear distintivos y si es posible mejoras.
- Etapa 2: Diseño del hardware base para el proyecto, configurando los módulos necesarios. Utilizando simulaciones, diagramas de bloques y esquemáticos.
- Etapa 3: Pruebas del CORE WAVEFORMGEN/NCO simulaciones y reales.
- Etapa 4: Pruebas realizadas en protoboard.
- Etapa 5: Construcción de PCB.
- Etapa 6: Ensamble de PCB.
- Etapa 7: Pruebas en el PCB.
- Etapa 8: Implementación del software utilizando el sistema operativo de tiempo real uC/OS II.
- Etapa 9: Uso de librerías gráficas, diseño del GUI e integración con uC/OS II para realizar la interacción con el usuario.
- Etapa 10: Pruebas finales en ambiente de producción integración hardware y software del sistema.

- Etapa 11: Pruebas finales en laboratorio realizando lecturas con el osciloscopio.

CAPÍTULO 2

2. MARCO TEÓRICO

Este capítulo contiene los fundamentos teóricos del hardware y software utilizado para el desarrollo del proyecto, además se explica el funcionamiento de la tecnología DDS que es vital para la generación de las señales.

2.1 PLATAFORMA TECNOLÓGICA SoC (“System-on-Chip”)

Se basa en la idea de integrar todos los componentes de computación u otros sistemas electrónicos, en un solo chip. Esta tecnología integra un procesador, módulos de memoria, periféricos de E/S y aceleradores de hardware personalizado.

Debido a su óptimo rendimiento, la miniaturización, menor consumo de energía y bajo costo, lo convierten hoy en día en la principal herramienta para los diseñadores de microprocesadores y circuitos electrónicos complejos, tiene como su principal aplicación a los **SISTEMAS EMBEBIDOS.**

El diseño de estos sistemas puede estar basado en circuitos de señal digital, señal analógica o señales mixtas.[1]

Figura 2.1 (SoC) System-on-Chip [2]

2.2 SISTEMAS EMBEBIDOS

Sistema Embebido es un tipo especial de sistema informático diseñado para realizar una o algunas tareas específicas, es decir se trata de un sistema cuyo hardware y software se diseñan específicamente.

El término embebido se refiere al hecho de que la microcomputadora es encapsulada en un solo circuito, están compuestos por microprocesadores, memorias, entradas y salidas a periféricos y un programa de aplicación dedicado.

Figura 2. 2 Sistemas Embebidos [3]

El diseño de un producto que incorpora sistemas embebidos generalmente está orientado a minimizar los costos, maximizar la confiabilidad, fiabilidad y flexibilidad. Ya que a la hora de realizar alguna

modificación resulta más sencillo modificar unas líneas de código al software del sistema que reemplazar todo el circuito integrado.

Un uso muy común de estos sistemas especiales es en los sistemas de tiempo real, es decir son aquellos sistemas en los que el control del tiempo es vital para el correcto funcionamiento. Los sistemas en tiempo real necesitan realizar ciertas operaciones o cálculos en un límite de tiempo. Donde ese límite de tiempo resulta crucial.

Los sistemas embebidos abarcan el 100% de la producción mundial de microprocesadores; son sistemas de hardware y software, algunos ejemplos de sistemas embebidos son los sistemas de información integrado en automóviles, barcos, trenes o aviones, y controladores de procesos en sistemas de producción industrial.

Figura 2. 3 Aplicaciones de los Sistemas Embebidos [4]

2.3 HARDWARE Y SOFTWARE DE UN SISTEMA

El Hardware de un Sistema Embebido consta en uno o más procesadores programables que permiten ejecutar el software de una aplicación, realizar tareas tales como automatización, video, procesamiento de señales, etc.

Los dispositivos de lógica programable modernos, especialmente los dispositivos FPGA, permiten la implementación de diferentes arquitecturas de procesamiento en hardware. La capacidad de las FPGA para la implementación de hardware permite la implementación de todo el sistema digital en un solo chip, lo que se denomina Sistema-en-Chip-Programable (SOPC, System-on-Programmable-Chip).

El Software o aplicación principal, es aquel que se encarga de realizar una tarea específica que en ocasiones dependiendo de la aplicación pueden funcionar en tiempo real, es decir realizar múltiples tareas o múltiples procesos permitiendo así darle un óptimo funcionamiento al sistema.

El proyecto utiliza la tarjeta de desarrollo DE0-Nano de altera, que cuenta con una FPGA de la familia Cyclone IVE, memorias, periféricos E/S, también la posibilidad de configurar un procesador embebido NIOS II que nos permiten desarrollar proyectos de electrónica digital complejo con todos los beneficios que tienen los sistemas embebidos.

2.3.1 TARJETA DE DESARROLLO DE0-NANO

La tarjeta DE0-Nano de Altera Cyclone IVE FPGA, es una tarjeta de desarrollo académica, portable para diseño de proyectos como robots, equipos móviles, dispositivos de electrónica digital.

Para proveer la máxima flexibilidad para el usuario, todas las conexiones se realizan a través del dispositivo Cyclone IV FPGA, de esta forma el usuario puede configurar la FPGA para implementar cualquier diseño.

Figura 2. 4 Tarjeta de Desarrollo DE0-NANO [5]

Las características principales de la tarjeta son:

- FPGA Altera Cyclone IV EP4CE2217C6N
  - 22.320 Elementos lógicos
  - 594-kb Memoria Embebida
  - 66 Embebidos 18x18 multiplicadores
  - 4 PLLs
  - 153 I/O pines
- Elemento de Configuración
  - USB-Blaster para la programación
  - Dispositivo EPCS64
- Pines de Expansión
  - 2 puertos de expansión GPIO de 40 pines provee 72 E/S pines
  - 2 pines de alimentación de 5V
  - 2 pines de alimentación de 3.3V
  - 4 pines de tierra
- Dispositivos de Memoria
  - 32 MB SDRAM
  - 2 kb I2C EEPROM
- Interfaces de Entrada y Salida
  - 8 LEDs
  - 2 pulsadores sin rebote
  - 4 interruptores (DIP)

- G-Sensor
  - ADI ADXL345, acelerómetro de 3 ejes con alta resolución (13 bit)
- Convertidor A/D
  - NS ADC128S022, 8 canales, 12 bit convertidor A/D; 50 a 200 [Ksps]
- Reloj del Sistema
  - Oscilador de 50 MHZ
- Fuente de alimentación
  - Puerto USB Tipo mini-AB (5V)
  - 2 pines de alimentación externa (3.6 – 5.7 V)

2.3.2 FPGA CYCLONE IV EP4C22

FPGA es un dispositivo lógico que contiene un arreglo de celdas lógicas programables e interconexiones. Las celdas lógicas pueden ser controladas por el diseñador lo que nos da la ventaja de una gran flexibilidad en el diseño de sistemas lógicos simples o complejos.

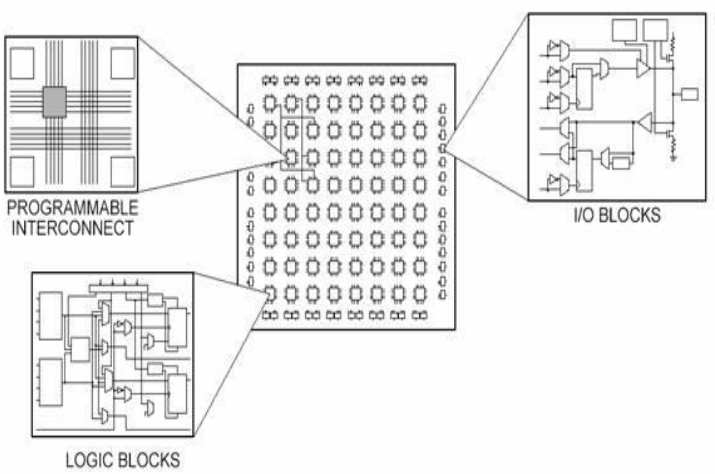


Figura 2. 5 Arquitectura FPGA [6]

Los FPGA se dividen en reprogramables (Basados en SRAM) y no reprogramables (One-Time-Programmable), los sistemas reprogramables dan una gran facilidad al diseñador de reprogramar el circuito una gran cantidad de veces, lo que permite corregir errores de diseño iníciales o proveer características nuevas o actualizaciones sin necesidad de cambiar el hardware. [7]

Los dispositivos FPGA modernos ofrecen una muy alta capacidad de implementación digital, incluyendo el desarrollo de sistemas de procesamiento y de comunicaciones digitales completos en un solo de estos dispositivos

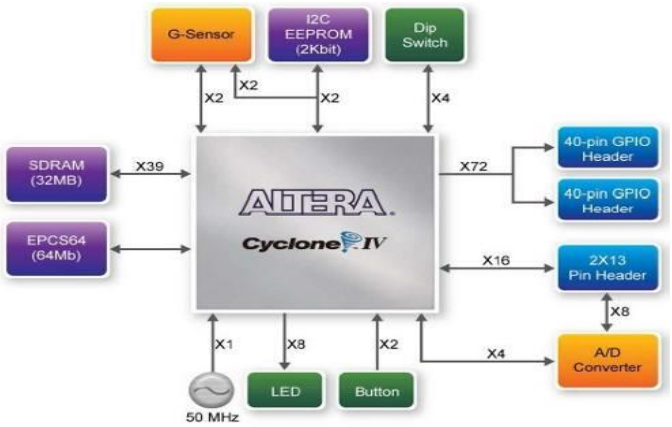


Figura 2. 6 Diagrama de bloque de la DE0-NANO [8]

El Cyclone IVE es un dispositivo semiconductor que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada en el instante mediante un lenguaje de descripción de hardware. La lógica programable puede realizar funciones sencillas hasta sistemas complejos, la interface JTAG es la que me permite realizar la programación en la FPGA. Requieren sólo dos fuentes de alimentación para la operación, lo que simplifica la distribución de energía, espacio de la placa, y el tiempo de diseño. Además la flexibilidad de la arquitectura de reloj transceptor le permite implementar múltiples protocolos de comunicación entre sus componentes, lo que nos permiten diseñar dispositivos a bajo costo

2.3.3 PROCESADOR NIOS II

Es un procesador embebido de 32 bits de propósito general del tipo ***RISC***, está diseñado específicamente para ser usado por medio de HDL en las FPGAs de la compañía ALTERA.

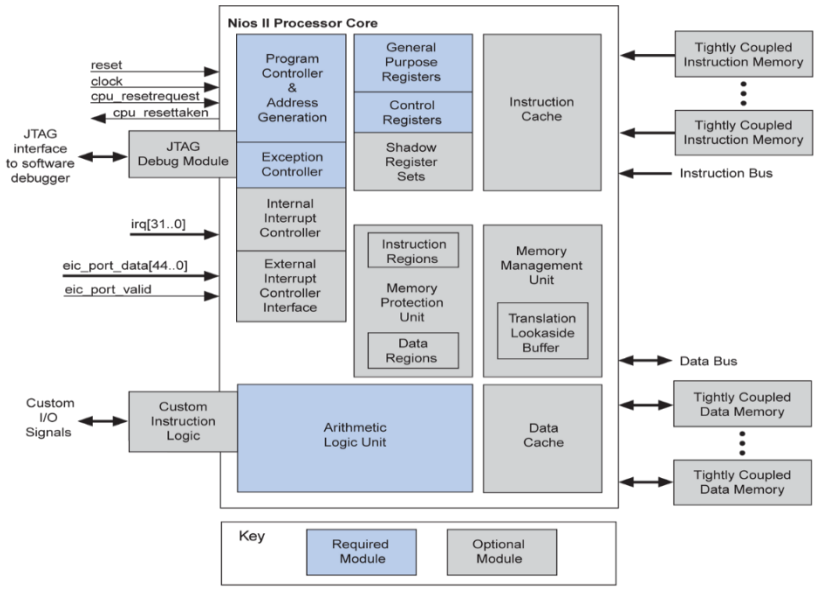


Figura 2. 7 Arquitectura general del procesador NIOD II [9]

La compañía de altera provee de tres versiones configurables del procesador NIOS II que el diseñador puede seleccionar según sus requerimientos:

Tabla 1 Versiones del procesador NIOS II

VERSIONES DE NIOS II	***DESCRIPCIÓN***
NIOS II / f	*Es un procesador rápido, el cual puede cumplir funciones de manera paralela y puede ejecutar líneas de código utilizando Pipeline, su arquitectura utiliza módulos hardware embebido para agilizar el procesamiento y convierta al Nios/f en un procesador para tareas de alto rendimiento.*
NIOS II / s	*Es un procesador estándar, el cual puede cumplir funciones complejas, para esto se añaden a su arquitectura más LEs que ayudan a que su procesamiento sea mucho más rápido, ideal para desempeñar tareas que requieran mucho más procesamiento, como interfaces gráficas de usuario.*
NIOS II / e	*Es un procesador de tipo económico que sacrifica elementos lógicos (LEs) por rendimiento, es el procesador más lento de los tres existentes, ideal para ser implementado en tareas que no necesiten altas velocidades de procesamiento.*

Todas las versiones de Nios II soportan un direccionamiento de memoria de al menos 2 GB, lógica para el soporte de depuración de software, la incorporación de instrucciones personalizadas y

la posibilidad de implementación en hardware a partir de funciones programadas en software.

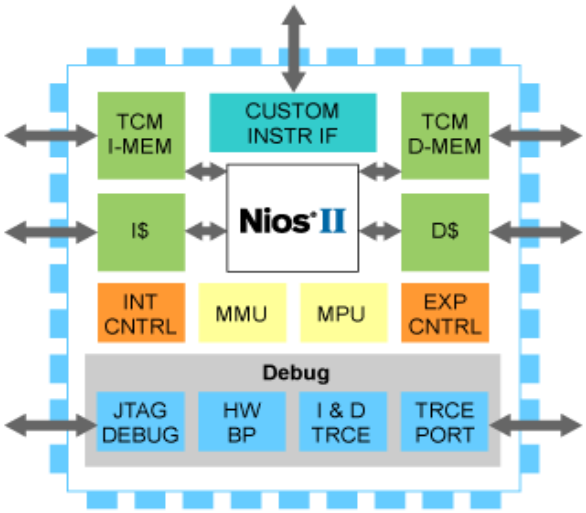


Figura 2. 8 Procesador NIOS II [10]

Este procesador puede adaptarse a sistemas de hardware reconfigurables que requieran un bajo o alto grado de rendimiento. Una de las ventajas es que al ser configurable el diseñador puede agregar o quitar periféricos sin necesidad de hacer cambios en el hardware, optimizando de esta forma el uso de recursos y dinero ya que el sistema se adapta a las necesidades específicas del usuario y el usuario no tiene por qué conformarse con el sistema o desperdiciar recursos de manera innecesaria.

2.3.4 QUARTUS II Y QSYS

QUARTUS II es un software de la compañía Altera que permite realizar el análisis e implementación de síntesis de diseños de hardware en dispositivos configurables sea estos (FPGA, CPLD).

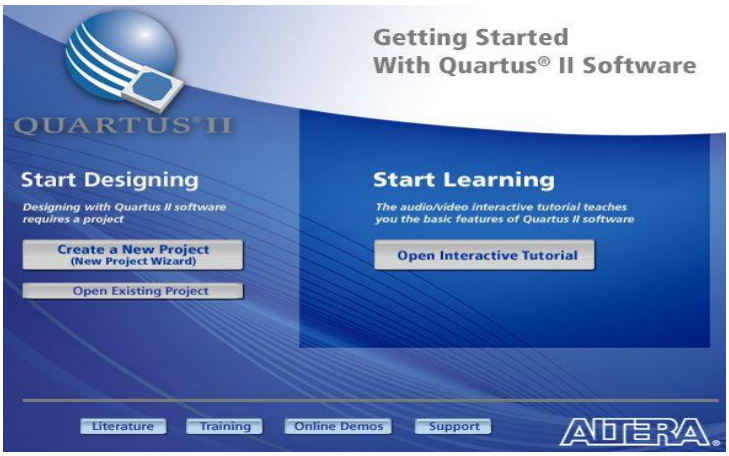


Figura 2. 9 Pantalla de bienvenida de Quartus II

QUARTUS II nos permite compilar los diseños, realizar análisis temporales, examinar diagramas RTL, simular los diseños digitales con ModelSim presentamos a continuación la descripción del flujo de diseño que nos brinda este software:

Tabla 2 Flujo de diseño en Quartus II

ETAPAS DEL PROCESO DE DISEÑO (QUARTUS II)	***DESCRIPCIÓN***
DISEÑO	*Abarca la descripción del hardware a implementar mediante la utilización de lenguajes de descripción de hardware, esquemáticos, gráficos, etc.*
SÍNTESIS	*Permite indicar las características del dispositivo a utilizar para generar un archivo que expresa el diseño en función de los recursos disponibles en dicho dispositivo.*
PLACE AND ROUTE	*Especifica los recursos de dispositivos que se van a utilizar y la manera en que éstos se interconectarán.*
ANÁLISIS TEMPORAL	*El software analiza si las frecuencias especificadas por el diseñador son satisfechas por la implementación sugerida.*
SIMULACIÓN	*Permite que el diseñador proponga diferentes escenarios en un ambiente virtual para comprobar el comportamiento del desarrollo ante dichos escenarios. La simulación permite la búsqueda de errores de diseño*
PROGRAMACIÓN Y CONFIGURACIÓN DEL DISPOSITIVO	*Esta es la etapa final del proceso que permite transferir desde la computadora la configuración del dispositivo programable. La configuración es almacenada como un archivo en la computadora que contiene toda la información necesaria para configurar un dispositivo lógico y que este se comporte como el diseñador especificó en la etapa de diseño.*

El objetivo del software de desarrollo es poder compilar un archivo de descripción de hardware (ya sea en forma de esquemático o archivo de descripción de hardware) a un archivo

que permita la configuración en la FPGA. Quartus II utiliza el concepto de proyecto para contener toda la información referente a un determinado diseño. Un proyecto es definido como todo el conjunto de archivos de diseño y de bibliotecas requeridas para crear un archivo de configuración del dispositivo FPGA.

Una de las herramientas adicionales que pueden incorporarse al ambiente de Quartus II es **Qsys**, el cual permite el diseño de sistemas basados en *soft-processors* y, en especial el procesador NIOS II. [11]

QSYS es una herramienta de desarrollo de Altera, poderosa y versátil para la creación de sistemas embebidos, prosee las siguientes características:

- Interfaz GUI intuitiva que permite realizar las conexiones entre los diferentes bloques, conectando muchas señales y buses juntos, con la interfaz realiza un listado y ordenamiento de los componentes del sistema, cada componente puede tener su propia interfaz de configuración, crea un archivo .**sopcinfo** con la descripción del sistema.

- Rápido para el desarrollo con soporte de diseño jerárquico, que permite realizar diseños digitales escalables.
- Posee una verificación rápida automática con el generador testbench.

En principios Qsys permitía el diseño de sistemas basados en el procesador de Altera NIOS II, aunque también puede ser utilizado tanto para la creación de sistemas con uno o más de estos procesadores, con procesadores de otros fabricantes o diseños que no poseen ningún procesador.

Realiza de manera óptima la integración de componentes de hardware en un sistema mayor, genera un archivo HDL que contiene todos los componentes y sus interconexiones.

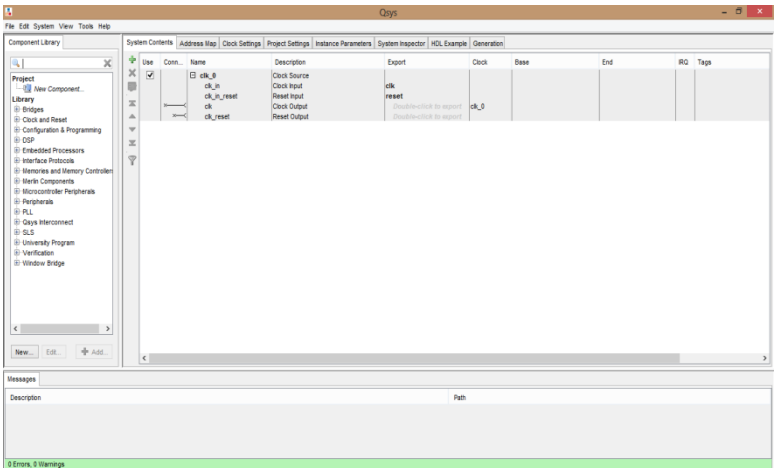


Figura 2. 10 Pantalla de bienvenida de Qsys

2.3.5 PROGRAMA NIOS II SBT

Nios II Software Build Tools de Eclipse es una herramienta de desarrollo de software embebido orientado al procesador NIOS II, la plataforma está totalmente integrada con Qsys lo que permite automáticamente crear el proyecto del paquete de soporte (BSP) incluye herramientas propietarias de código libre para la creación de proyectos, cuenta con una interfaz gráfica de usuario basada en Eclipse compila código ANSI-C estándar y lenguaje C/C++.[12]

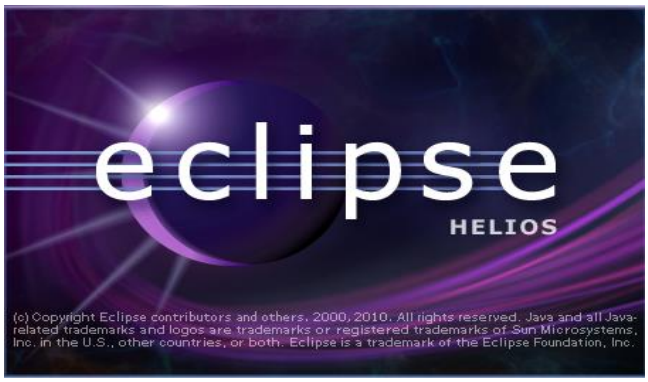


Figura 2. 11 Pantalla de inicio de NIOS II SBT

NIOS II SBT permite leer el archivo .sopcinfo, que contiene la información del diseño de hardware para en base a esto generar

una aplicación de software que me permita desarrollar las funcionalidades del sistema.

Todas las herramientas de desarrollo de sistemas embebidos se basan en la necesidad de dar un soporte de software que le permita abstraerse en lo posible de la arquitectura de hardware y, al mismo tiempo, este soporte requiera la menor cantidad de recursos del sistema.

Es por eso que se proponen alternativas para la implementación de capas de abstracción de hardware (HAL, Hardware Abstraction Layer), que permite enlazar el hardware (Quartus II – Qsys) y el software (IDE NIOS II SBT).

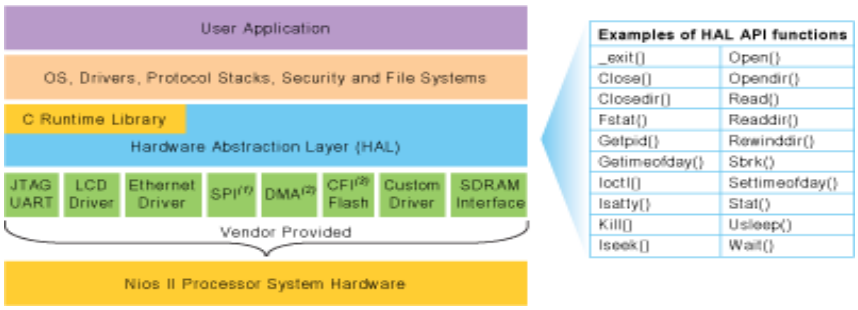


Figura 2.12 NIOS II HAL[13]

HAL presenta beneficios muy importantes al momento de diseñar una aplicación:

Sencillez: Facilita la reutilización de código entre diseños, permite la integración directa de los controladores de los periféricos del SOPC.

Confiabilidad: Permite la configuración de los diferentes periféricos mediante estructuras definidas y mejora la consistencia del software mediante el acceso a los periféricos.

Estandarización: Permite la realización del código mediante la utilización de una interfaz estándar.

2.3.6 CARACTERÍSTICAS Y ARQUITECTURA DEL SISTEMA

En la Figura 2.11 muestra un diagrama de bloques simplificado del hardware del sistema.

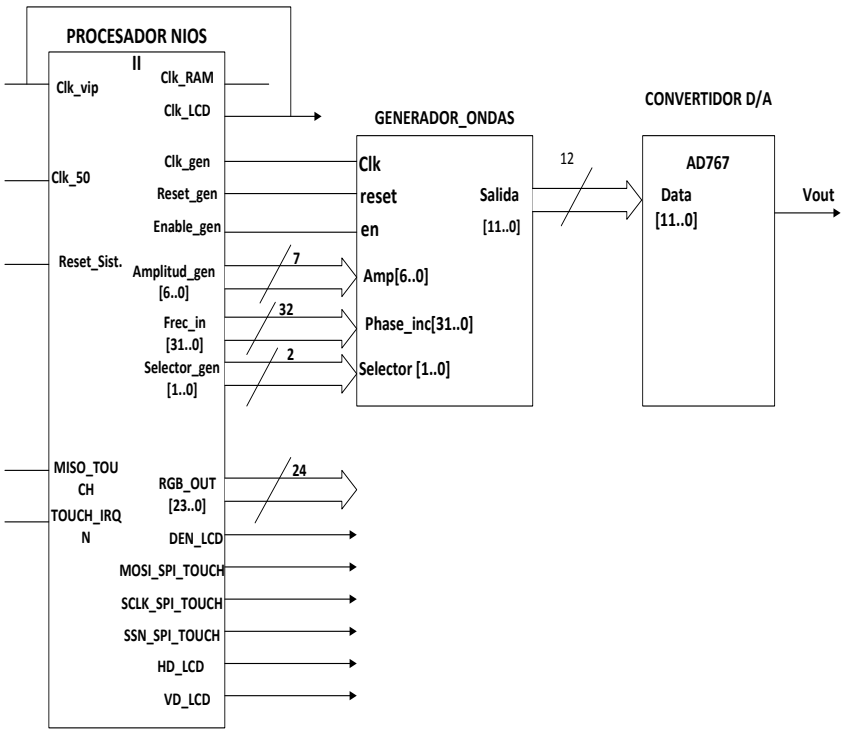


Figura 2.13 Arquitectura del Proyecto Generador de Señales

EL bloque inicial corresponde a la máquina principal del sistema, el hardware diseñado con el PROCESADOR NIOS II específicamente con las memorias, puertos requeridos para la comunicación serial SPI de la pantalla LCD-Touch. Además se configuran los puertos PIO de la máquina principal necesarios para interactuar con el bloque WAVEFORM_GEN/NCO de forma externa, siguiendo con el diagrama tenemos un bloque SELECTOR que determina la señal seleccionada cabe

mencionar que ese selector está controlado por una señal que sale de la máquina principal. Luego de esta fase tenemos 12 bits de salida con una forma de onda discreta que ingresará al DAC para realizar la conversión a voltaje analógico y así obtener el proceso completo del generador de señales.

Cabe recalcar que la interfaz gráfica y demás eventos que permiten realizar la interacción del generador de señales con los usuarios será realizada por software con la ayuda del IDE NIOS II SBT el cual permite realizar la programación del sistema embebido. En la sección del software el proyecto trabaja con el sistema operativo en tiempo real “uC/OS II”.

2.4 TECNOLOGÍA DDS

Síntesis Digital Directa (DDS) es un método muy utilizado para generar formas de onda o señales debido a su excelente resolución de frecuencia, flexibilidad, simetría, modulación y capacidad de almacenamiento que nos permite emular por medios digitales una señal analógica. Esta técnica produce frecuencias digitales o formas de onda análoga desde una fuente de reloj fija.

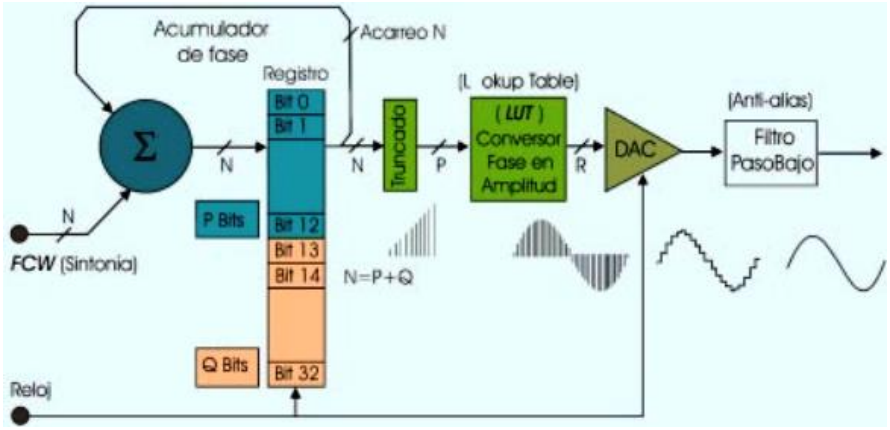


Figura 2.14 Esquema básico de DDS [14]

Es un sistema digital que consta de los siguientes componentes:

Acumulador de Fase*:* Es un sumador digital que cada pulso activo de reloj suma el valor de la fase ***Fcw*** con el valor acumulado previamente en A y lo vuelve a almacenar en A para producir la salida Q [15]

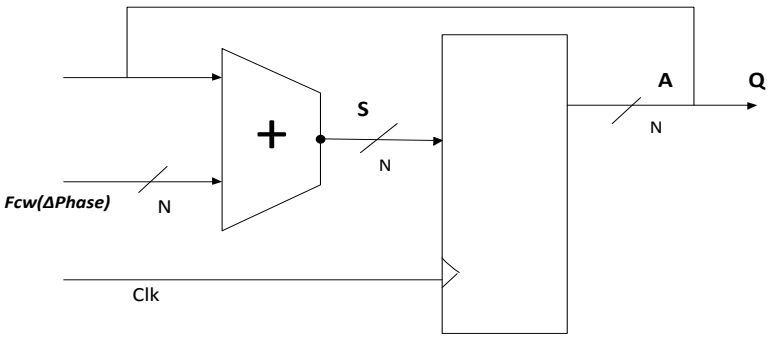


Figura 2.15 Acumulador de Fase

$$F(out) = \frac{Fcw * Fclk}{2^N} \qquad \textbf{(2.1)}$$

LUT ("Lookup Table") o Tabla de Consulta: Es la estrategia más simple para generar ondas sinoidales, el acumulador de fase genera una señal incremental semejante a un diente de sierra, de la frecuencia deseada; pero para generar señales sinusoidales es necesario convertir los valores de dicha señal en valores de una onda seno/coseno. Esta tabla de consulta tiene la estructura de un codificador, con la diferencia de que una tabla de consulta suele ser mucho más grande, estos valores son almacenado por lo general en una memoria "ROM". Los vectores digitales a la salida de la LUT representan la amplitud de una señal sinusoidal para los correspondientes valores de fase.

DAC (Digital - Analógico)*:* Convertidor digital – analógico para realizar la conversión de la onda muestreada, a la salida de la memoria, en una onda escalonada analógica. [15]

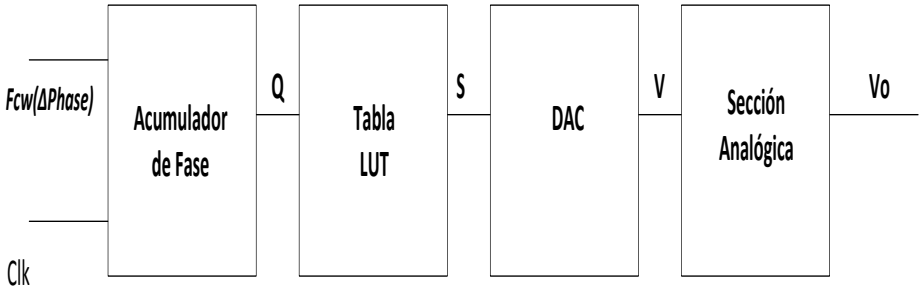


Figura 2.16 Diagrama de bloques de un DDS

Como se puede ver en la Figura 2.16 el DDS trabaja en base a la frecuencia del sistema que lo determina la señal Clk, produce una rampa (diente de sierra) digital Q con una frecuencia proporcional Fcw. La LUT convierte o codifica la señal Q en una forma de onda arbitraria S.

El DAC es el que convierte la señal digital S en una señal analógica que en la sección analógica se encarga de realizar el proceso final para obtener la forma de onda deseada [15]

2.5 HARDWARE: IP CORE WAVEFORM_GEN/NCO DE OPENCORES

OpenCores es la comunidad de hardware de código abierto más grande del mundo que desarrolla hardware libre núcleos IP a través de la automatización de diseño electrónico, similar al software libre. [16]

Figura 2.17 Página de inicio de OpenCores.org

Esta compañía con la finalidad de realizar diseños digitales modulares tiene disponible IP CORES de libre acceso, en la sección ***Project*** y la categoría ***DSP Core*** encontraremos el "***NCO / Periodic WaveformGenerator".***

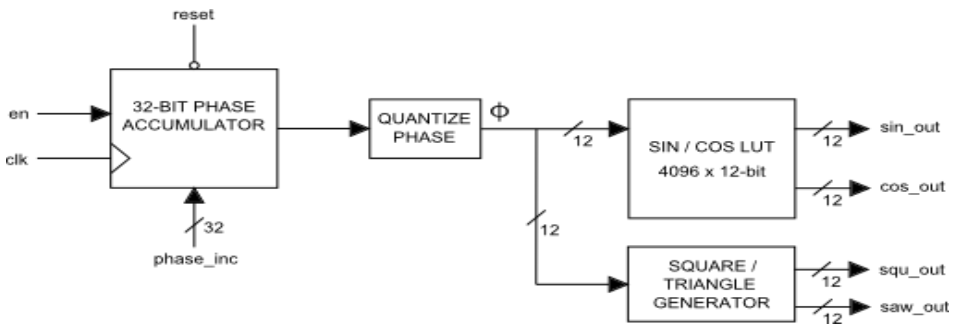


Figura 2.18 NCO Arquitectura [17]

Las características principales del Core son:

- 12 bits de salida de muestreo con signo.
- 32 bits del acumulador de fase (Palabra Digital de sintonía ***"phase_inc"***).
- Resolución de Fase está dado por: $^{2\pi}/_{2^{12}}$
- La resolución de la Frecuencia: $\frac{Fs}{2^{32}}$ (Fs = Frecuencia de muestreo).
- 70 dB SNR
- 70 dB SFDR
- Salida simultáneo de formas de onda Seno, Coseno, Cuadrada y Diente de sierra (Rampa).
- 2 ciclos de reloj retraso.
- Tasa de muestreo de 500Mhz

Tabla 3 Descripción de Puertos WAVEFORMGEN

Pin	*E/S*	*Descripción*	*Estado Activo*
Clk	*Entrada*	*Clock de Muestra*	*Flanco Positivo*
Reset	*Entrada*	*ResetAsynchronous*	*Bajo*
En	*Entrada*	*Clock habilitador*	*Alto*

phase_inc [31:0]	*Entrada*	*Incremento de fase de 32 bits sin signo (controla la frecuencia de salida de la forma de onda)*	*Dato*
sin_out[11:0]	*Salida*	*Onda Seno Salida (16 bits número con signo)*	*Dato*
cos_out[11:0]	*Salida*	*Onda Coseno Salida (16 bits número con signo)*	*Dato*
squ_out[11:0]	*Salida*	*Onda Cuadrada Salida (16 bits número con signo)*	*Dato*
saw_out[11:0]	*Salida*	*Onda Diente Sierra Salida (16 bits número con signo)*	*Dato*

2.5.1 ANÁLISIS DEL CORE NCO

El WAVEFORM_GEN/NCO es un Oscilador Numéricamente Controlado de 12 bits con salida simultánea de formas de onda Seno, Coseno, cuadrada y diente de sierra.

Cada flanco de reloj positivo la señal “phase_inc” (32bits) del acumulador de fase es incrementado. Esta fase es cuantizada a 12 bits, es decir de los 32 bits del acumulador toma los 12 bits MSB para enviar a la dirección de la tabla “Look-up” LUT la cual convierte la fase en la forma de onda deseada

La Tabla LUT esta implementada con un arreglo de 4096 x 12 bits de muestras sobre el rango de [0, 2π]. Todas las salidas de valores son de 12 bits con signo.

La frecuencia de salida de la forma de onda es controlada por el incremento de fase. Un cambio en el incremento de fase durante el funcionamiento normal del circuito afectará a la salida de la forma de onda en 2 clocks de muestreo más tarde.

El incremento de fase lo obtenemos de la siguiente ecuación:

$$\emptyset(inc) = {}^{(Fout * 2^{32})}/_{Fs} + 0.5 \qquad \textbf{(2.2)}$$

- Fout es la frecuencia de salida de la forma de onda deseada.
- Fs es la frecuencia de muestreo (fclk).

EL Mínimo y máximo de las frecuencias del NCO son determinadas por las siguientes ecuaciones:

$$F(min) = \frac{Fs}{2^{32}} \qquad \textbf{(2.3)}$$

$$F(máx) = \frac{Fs}{2} \qquad \textbf{(2.4)}$$

2.5.2 VENTAJAS Y USO DEL CORE WAVEFORM_GEN/NCO

El Core nos permite realizar el diseño modular, entre las ventajas que nos da tenemos que nos ayuda en la implementación de la tecnología DDS antes mencionada que es óptima en el desarrollo de un generador de señales.

Así también tenemos que con el Core podemos realizar:

- Osciladores Digitales
- Modulación Digitales (ASK, FSK, PSK, QAM)
- Generador de Señales
- Generador de señales complejas de Cuadratura

2.6 CONVERTIDOR DIGITAL/ANALÓGICO AD767

Es necesario para convertir los niveles discretos obtenidos a la salida de la tabla LUT utilizar un convertidor Digital – Analógico (DAC), estos dispositivos contienen normalmente una red resistiva divisora de tensión, que tiene una tensión de referencia estable y fija como entrada. En nuestro caso escogimos el convertidor AD767 que posee las siguientes características:

- 12 bit D/A – con alta estabilidad
- Conversión Paralela
- Garantizada para funcionamiento con voltaje: +12, -12; +15,-15 [V]
- Tiempo Máximo de conversión: 4 *u*s
- TTL/5 V CMOS

Figura 2.19 Convertidor DAC AD767 DIP [18]

EL AD767 posee alta estabilidad referenciada por un Zener. Este convertidor utiliza 12 bits de precisión, alta velocidad de conversión.

Es un DAC de voltaje completo con voltaje de referencia y LATCHES (circuito electrónico usado para almacenar bits de información), en un simple CI chip.

La entrada LATCH responde a escribir anchos de pulso tan pequeños como 40ns asegurando una interfaz directa con los microprocesadores más rápidos de la industria. [19]

2.7 MÓDULO DE LA PANTALLA LCD TOUCH

El tipo de pantalla usado en el proyecto es TFT-LCD (“Thin Film Transistor-Liquid Crystal Display”) que significa “Pantalla de cristal líquido de transistores de película fina”, con una resolución de 480x272 pixeles y una dimensión de 4.3 pulgadas medido diagonalmente, la pantalla junto con la membrana Touch resistiva cumple la funcionalidad de la interface de usuario para el generador de funciones.

La profundidad de color de la pantalla o bits por pixel es de 24 bits, 8bits para cada uno de los tres tonos primarios de rojo, verde y azul, permitiendo así que cada pixel pueda representar 2^{24} o lo que es lo mismo 16.7 millones de colores. La transferencia desde la tarjeta DE0-nano a la pantalla de forma paralela, es decir, se envía al mismo tiempo los 24 bits de color de cada pixel en cada flanco de reloj, la pantalla utiliza una velocidad de reloj de 9 Mhz.

La membrana touch resistiva tiene un controlador modelo XPT2046 la cual utiliza comunicación SPI con la tarjeta DE0-nano.

A continuación se presenta una tabla con la asignación de pines de la pantalla táctil y su función:

Tabla 4 Pines de la pantalla TFT-LCD

Pin#	SYMBOLO	DESCRIPCION	TIPO	FUNCION
1	IRQ	Interruptor de membrana táctil	output	Envía señal en bajo cuando detecta que la membrana táctil ha sido presionada
2	5V	Alimentación 5V	Input	Alimentación de 5V DC
3	MOSI	SPI data input de la membrana táctil	Input	EL NIOS II envía al controlador SPI de la membrana táctil una secuencia de bits que indica al controlador el tipo de información que debe responder, en este caso las coordenadas de donde se presiona la pantalla. Se conecta a SPI MOSI en el módulo SPI dentro del NIOS II
4	MISO	SPI data output de la membrana táctil	output	Envía información “cruda” de las coordenadas donde se presionó la pantalla. Se conecta a SPI MISO en el módulo SPI dentro del NIOS II.
5	SCK	SPI Clock de la membrana táctil	Input	Frecuencia de reloj para la comunicación SPI entre la membrana táctil y el sistema NIOS II. Se conecta a SPI SCK en el módulo SPI dentro de NIOS II.
6	SSEL	Touchscreen chip select	Input	Activo Bajo

7	PWM	Ajuste de Brillo	Input	La pantalla recibe una señal PWM con voltaje pico de 3.3V para ajustar el brillo. También se la puede conectar directamente a 3.3V y tener un brillo de la pantalla constante.
8	GND	Tierra	Input	Tierra digital
9	BUSY	Membrana táctil ocupada	output	Señal en alto que envía el controlador de la membrana táctil, cuando éste se encuentra enviando o recibiendo información a través de los pines 3 y 4.
10	NC	Sin uso		Sin uso
11	R0	Datos de color	Input	8 bits de color para el tono rojo.
12	R1			
13	R2			
14	R3			
15	R4			
16	R5			
17	R6			
18	R7			
19	G0	Datos de color	input	8 bits de color para el tono verde.
20	G1			
21	G2			
22	G3			
23	G4			
24	G5			
25	G6			
26	G7			
27	B0	Datos de color	Input	8 bits de color para el tono azul.
28	B1			
29	B2			
30	B3			
31	B4			
32	B5			

33	B6			
34	B7			
35	DCLK	LCD CLOCK	Input	Señal de reloj para la pantalla
36	DSIP	NC		
37	HSYNC	Sincronización Horizontal	Input	Señal de sincronización horizontal de la pantalla
38	VSYNC	Sincronización Vertical	Input	Señal de sincronización vertical de la pantalla
39	DE	Control MODO SELECCIÓN	Input	DE = 0 : SYNC mode DE = 1 : DE mode
40	GND	Tierra	Input	Tierra Digital

2.8 SISTEMA OPERATIVO uC/OS II

El sistema operativo uC/OS-II es el acrónimo para “Micro-Controller Operating Systems - Versión 2”. Es un sistema operativo multitarea en tiempo real basado en prioridad para microprocesadores, microcontroladores y procesadores de señales digitales, está escrito principalmente en el lenguaje de programación “C”. Su uso es intencionado para sistemas embebidos.

2.8.1 FUNDAMENTO TEÓRICO DEL “uC/OS-II”

El sistema operativo uC/OS-II es un conjunto de bibliotecas que se añaden como parte del código de un sistema embebido. Tiene archivos de encabezado .h los cuales son necesarios configurar y depende del tipo de microprocesador o microcontrolador que se quiera usar.

Funciona administrando tareas (“Tasks”): cada tarea realiza una funcionalidad específica de la aplicación que se desea crear, la tarea se la crea de tal forma que se encuentre ejecutándose continuamente dentro de un ciclo infinito. Una tarea puede obtener información sobre si misma o sobre otra tarea de tal forma que una tarea puede suspender otra tarea o a sí misma, también puede cambiar la prioridad de otra tarea.

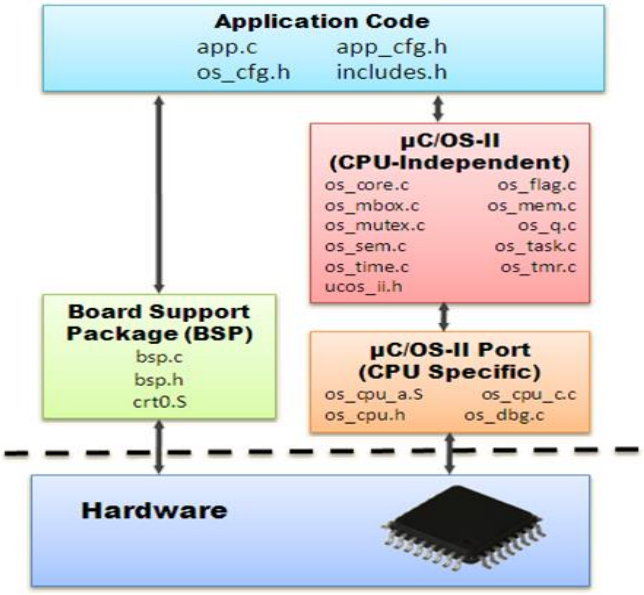


Figura 2.20 Características del uC/OS II [20]

A cada tarea se le asigna un espacio de memoria continua "Stack" donde se almacena la información de los datos que maneja.

El uC/OS-II solamente puede ejecutar una sola tarea a la vez, la forma en que maneja multitareas es poniendo a "dormir" las demás tareas, de tal forma que cada tarea tiene que esperar su turno para ejecutarse.

La forma en que se elige que tarea se ejecutará es mediante la prioridad que se le asigna

2.8.2 APLICACIONES Y VENTAJAS DEL "uC/OS-II"

EL uC/OS-II se lo puede aplicar en cualquier proyecto donde la aplicación tenga que realizar diferentes tareas a la vez.

Una de las principales ventajas del uso de uC/OS-II es la portabilidad, es decir, el código de la aplicación se la escribe en lenguaje C/C++ y uC/OS-II se encarga de la interacción de la aplicación con el hardware lo cual permite a la aplicación ser ejecutada en diversas plataformas sin ser necesario reescribir enteramente el código.

CAPÍTULO 3

3. DISEÑO E IMPLEMENTACIÓN

Este capítulo se hace una descripción completa del diseño del generador de señales y la forma en que se implementó. Como se indicó anteriormente las ventajas de las FPGAs nos permiten la implementación de nuevas tecnologías y lógica compleja.

El proyecto se basa en realizar un SOPC (Sistema-en-chip-programable) el cual posee el diseño de hardware y del software asociado.

Las herramientas necesarias para la implementación del proyecto son:

- DE0-NANO de Altera.
- Wave Share TFT LCD 4.3'' 480x272px Touch Screen.

- Convertidor D/A AD767 conversión en paralelo, 12 bits de precisión.
- Fuente Bipolar +12, -12 [v].
- Capacitor 20pF, 1 Trim 100 [Ω] aproximadamente.
- 1 Regulador 7805 de 5[V], 2 Capacitores 0.1 [uF].
- 1 conector BNC.
- 64 cables para protoboard hembra/hembra.
- 1 PC con software Quartus II 12.1, Eclipse - NIOS II SBT.

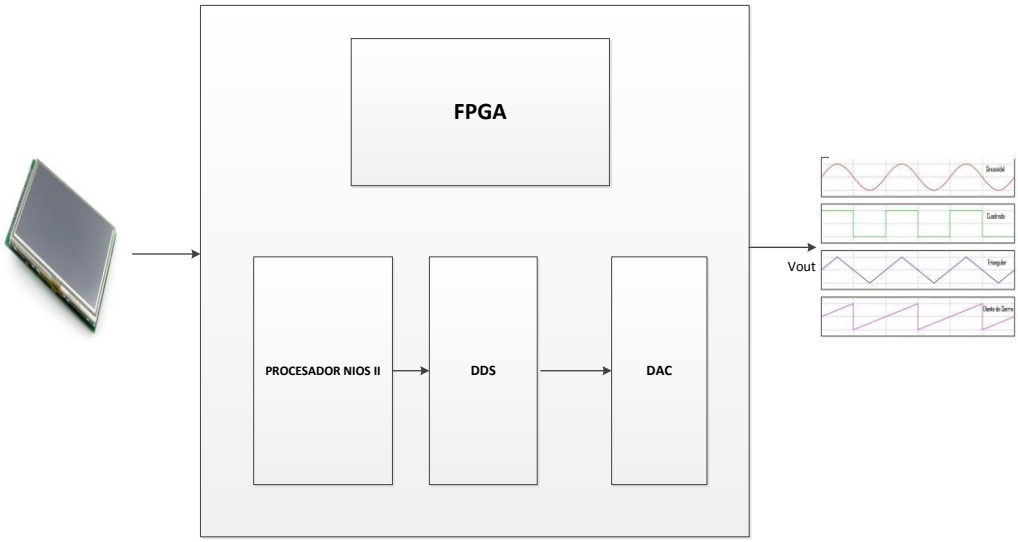


Figura 3.1 Diagrama de Bloques del proyecto generador de señales

En la Figura 3.1 presentamos el diagrama de bloque general, es decir los elementos principales que conforman el sistema.

3.1 ANÁLISIS Y JUSTIFICACIÓN DEL DISEÑO

Basándonos en equipos para laboratorios de electrónica, se consideró las siguientes características de un generador de funciones análogo:

- Los generadores de señales también son conocidos como sintetizadores de función o multifunción y generan distintas formas de onda como sinusoidal, cuadrada, triangular y rampa con precisión en rangos de frecuencia desde mHz hasta varios MHz. [21]
- El voltaje máximo pico a pico de estos generadores de funciones varían entre los 5 a 20 V.
- Es posible modificar el valor de frecuencia, amplitud.
- La forma por la cual se controla el generador de funciones es mediante botones, perillas y una pantalla LCD monocromática por lo cual se presenta la información de la onda a generar.

Con lo antes mencionado nos dimos cuenta que podíamos crear un generador de funciones digital que podría competir con las características de un generador de funciones análogo de laboratorio y a la vez bridar ciertas ventajas:

- La interfaz de usuario de nuestro generador de funciones es mediante una pantalla táctil, en la cual se puede modificar los parámetros de la función a generar de la misma forma que en un generador análogo.
- Se puede generar ondas sinusoidal, cuadrada, triangular y diente de sierra.
- Es posible generar ondas hasta 10 V pico-pico y variar la frecuencia desde 1 Hz hasta 550 KHz para el seno, 300Khz para la cuadrada y triangular y 180 KHz para la diente de sierra en incrementos de 1 Hz.
- A diferencia de un generador de funciones análogo, nuestro generador de funciones digital es compacto y portátil ya que pesa menos de 1 kg.

El costo de implementación de muestro generador de funciones es menos de $200 comparado con un generador de funciones de igual prestaciones que ronda por los $350 el más básico.

Un generador de funciones digital, utiliza un método de sintetización de frecuencias llamada DDS, la cual consiste en efectuar con una o más señales de frecuencia estable, operaciones matemáticas (suma, resta, multiplicación y divisiones) a fin de obtener en la salida una señal cuya frecuencia sea la deseada. Éste método nos da muchas ventajas sobre

otras configuraciones utilizadas para generar formas de onda, la principal de todas la estabilidad y amplio rango de frecuencias, por esta razón justificamos nuestra decisión de diseñar el proyecto basándonos en esta tecnología y para brindarle a los usuarios del equipo una interacción práctica. Además de la posibilidad de poder utilizar este proyecto de referencias para futuras mejoras del mismo, como por ejemplo agregar la funcionalidad de que además de ser un generador de señales, también sea un osciloscopio digital y visualizar por la pantalla las señales.

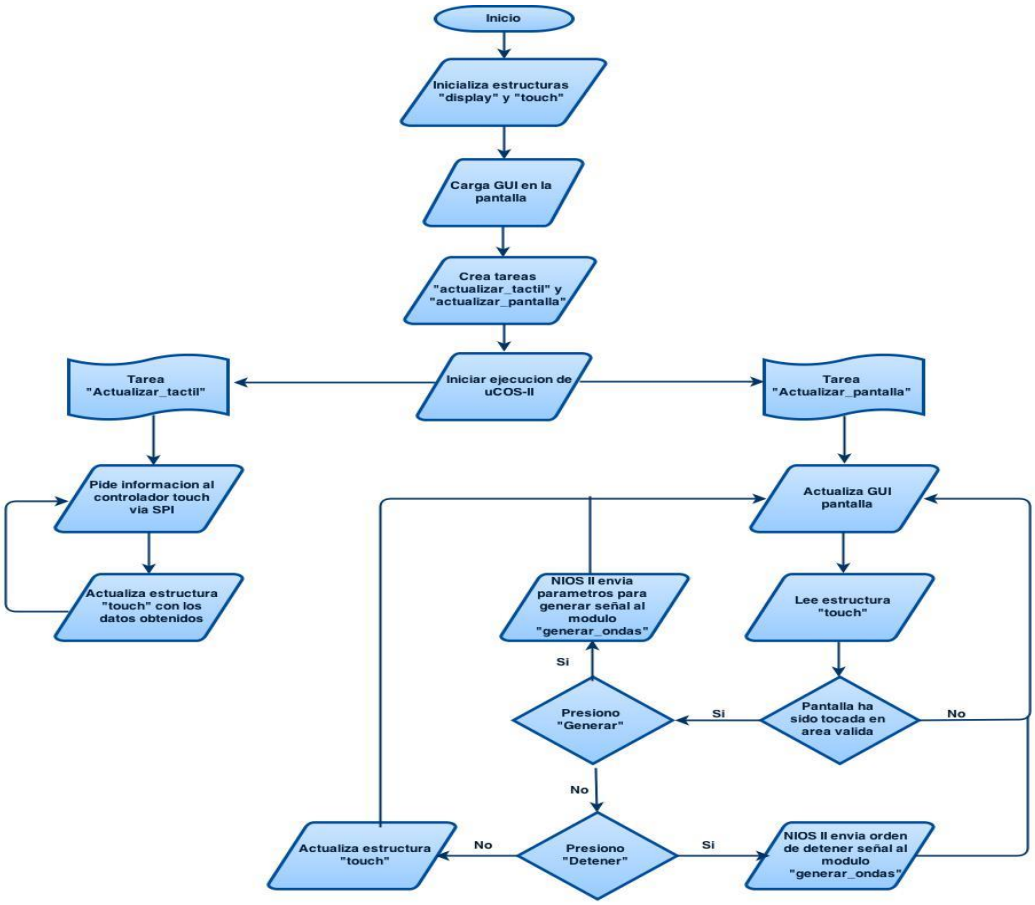


Figura 3.2 Diagrama de Flujo del Sistema

3.2 CARGAR LA INTERFAZ GRÁFICA E INGRESO DE DATOS SEGÚN EL MENÚ

Para el ingreso de los datos, se desarrolló una GUI presentada en una pantalla táctil TFT LCD 4.3'', es decir a través de un menú diseñado en la interfaz gráfica donde se visualizan los botones que permiten a los

usuarios interactuar y seleccionar los parámetros de la señales mediante la membrana táctil de la LCD. En el software, para implementar la interfaz, se utiliza la ayuda de las librerías gráficas de altera:

- simple_graphics.c y simple_graphics.h
- simple_text.c y simple_text.h
- draw_gimps.c y draw_gimps.h
- gimp_bmp.c y gimp_bmp.h
- alt_video_display.c y alt_video_display.h
- vip_fr.c y vip_fr.h

Los mismos que nos permiten utilizar las funciones necesarias para realizar dibujos de formas geométricas, botones, colocar imágenes en el espacio de memoria de video.

```
int main (void) {
/*Puntero que maneja la estructura VIP_FRAME_READER*/
VIP_FRAME_READER *display;
touch_control touch;
/* Se inicializa el HW del sistema */
display = VIPFR_Init (ALT_VIP_VFR_BASE, (void *) FR_FRAME_0,
(void *) FR_FRAME_0, FRAME_WIDTH,   FRAME_HEIGHT);
VIPFR_Go (display, TRUE);
/* Se activa el Frame de video */
VIPFR_ActiveDrawFrame (display);
vid_clean_screen (display, WHITE_24);
/* Procedimiento que dibuja de la interfaz gráfica */
dibuja_gui (&touch, display);
return 0; }
```

Figura 3.3 Algoritmo para iniciar la sección gráfica de la pantalla

El código presentado en la Figura 3.3 sigue el proceso para iniciar y realizar la activación del controlador Frame que maneja la sección gráfica de la pantalla, utilizando sus funciones principales como `VIPFR_Init (), VIPFR_ActiveDrawFrame (), etc.`

```
    void dibuja_gui (touch_control *touch, VIP_FRAME_READER
    *display) {

    VIPFR_ActiveDrawFrame (display);
    DispIMAGE((void*)display, 0, -8,&ESPOL);

     /* AMPLITUD */
     DispIMAGE((void*)display, 40, 47,&BTN_ATRAS);
     DispIMAGE((void*)display, 62, 40,&AMPLITUD);
     DispIMAGE((void*)display, 170, 47,&BTN_ADELANTE);

     /* FRECUENCIA */
     DispIMAGE((void*)display, 205, 47,&BTN_ATRAS);
     DispIMAGE((void*)display, 227, 40,&FRECUENCIA);
     DispIMAGE((void*)display, 332, 47,&BTN_ADELANTE);

     /* Procedimiento encargado de pintar los valores
     cargados inicialmente en la GUI */
     mostrar_valores (touch, display);

     /*En detalle el código en ANEXOS */
     (…)
}
```

Figura 3.4 Fragmento de código del procedimiento que dibuja el diseño de la GUI

En la Figura 3.4 se presenta parte del código encargado de realizar el dibujo del diseño de la GUI en la pantalla utilizando funciones gráficas como por ejemplo `DispIMAGE ()` y `vid_print_string ()`.

3.3 MOSTRAR LOS DATOS Y GENERA LA SEÑAL

En el proyecto se utiliza la pantalla táctil TFT LCD 4.3” Touch Screen para mostrar el menú e interactuar con el usuario, con el siguiente procedimiento se presenta los valores de amplitud y frecuencia.

```
/* Procedimiento que recibe los punteros touch_control. El display
posee la estructura de la gráfica*/
void mostrar_amplitud(touch_control *touch,
VIP_FRAME_READER *display){
char valor_pantalla[8];
snprintf(valor_pantalla,8,"%d", touch->datos.amplitud);
vid_draw_box(100,53,130,63,WHITE_24,DO_FILL,display);
vid_print_string(100, 53, SLATEBLUE_24, cour10_font,
display, valor_pantalla);
}

void mostrar_frecuencia(touch_control *touch,
VIP_FRAME_READER *display){
char valor_pantalla[8];
float frec_pantalla = (touch->datos.frecuencia);
if(touch->parametros.tipo_frec==UI_HZ)
snprintf(valor_pantalla,8,"%.1f",frec_pantalla);
else if(touch->parametros.tipo_frec==UI_KHZ)
snprintf(valor_pantalla,8,"%.4f",(frec_pantalla/1000));
else if(touch->parametros.tipo_frec==UI_MHZ)
snprintf(valor_pantalla,8,"%.4f",(frec_pantalla/1000000));
vid_draw_box(260,53,315,63,WHITE_24,DO_FILL,display);
vid_print_string(260, 53, SLATEBLUE_24, cour10_font,
display, valor_pantalla);}
```

Figura 3.5 Código que realiza el procedimiento para presentar y calcular los valores de amplitud y frecuencia

A medida que el usuario va interactuando con la pantalla, ésta a su vez se actualiza mostrando los nuevos valores seleccionados.

Luego de que el usuario está satisfecho con los parámetros de la señal que desea generar, para esto realiza el envío de esos valores al módulo NCO en hardware para proceder con la creación de la señal discreta y a su vez enviar la señal al convertidor DAC para finalmente tener la señal análoga.

```
#include<stdio.h>
#include"system.h"
#include"touch_handlers.h"
#include<math.h>

volatileint *Enable_GEN = (int *)ENABLE_GEN_BASE;
volatileint *ResetN = (int *) RESET_GEN_BASE;
volatileint *Phase = (int *) FRECUENCIA_IN_BASE;
volatileint *Amplitud = (int *)AMPLITUD_BASE;
volatileint *Select = (int *) SELECTOR_GEN_BASE;

void generarHardware (touch_control *touch) {
*ResetN = 1;
*Enable_GEN = 1;
*Phase = (int) (touch->datos->incrementado);
*Select = touch->datos->onda;
*Amplitud = (int) (touch->datos->amplitud);
}
```

Figura 3.6 Código del procedimiento que envía los parámetros al CORE WAVEFORMGEN/NCO

En esta ocasión la Figura 3.6 muestra el procedimiento para enviar los datos al hardware mediante los PIOs configurables, para habilitar el CORE ***"Generador_Ondas"***.

3.4 ESQUEMÁTICOS DEL CIRCUITO GENERADOR DE SEÑALES DIGITAL

En la siguiente sección se muestra los esquemáticos requeridos para el desarrollo del proyecto:

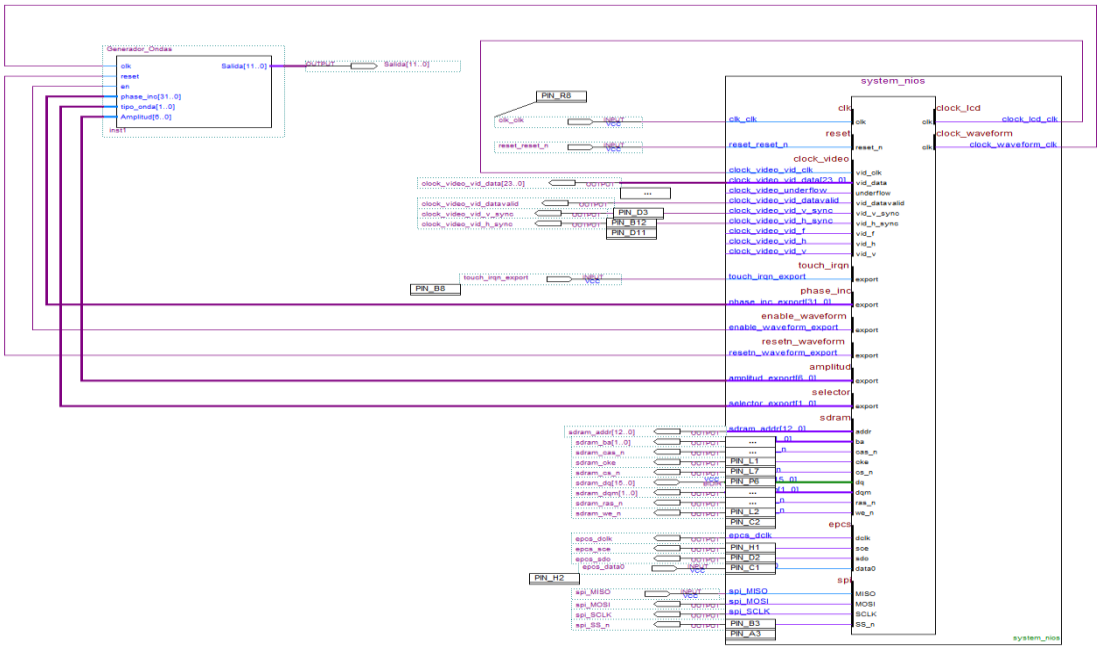


Figura 3.7 Diagrama esquemático del proyecto generador de señales

A continuación se presenta el esquemático de la placa PCB donde se tiene el bosquejo de la configuración del D/A AD767, los elementos necesarios para obtener las señales y otros componentes que se utilizan para polarizar el circuito del D/A como también la alimentación de la tarjeta DE0-NANO.

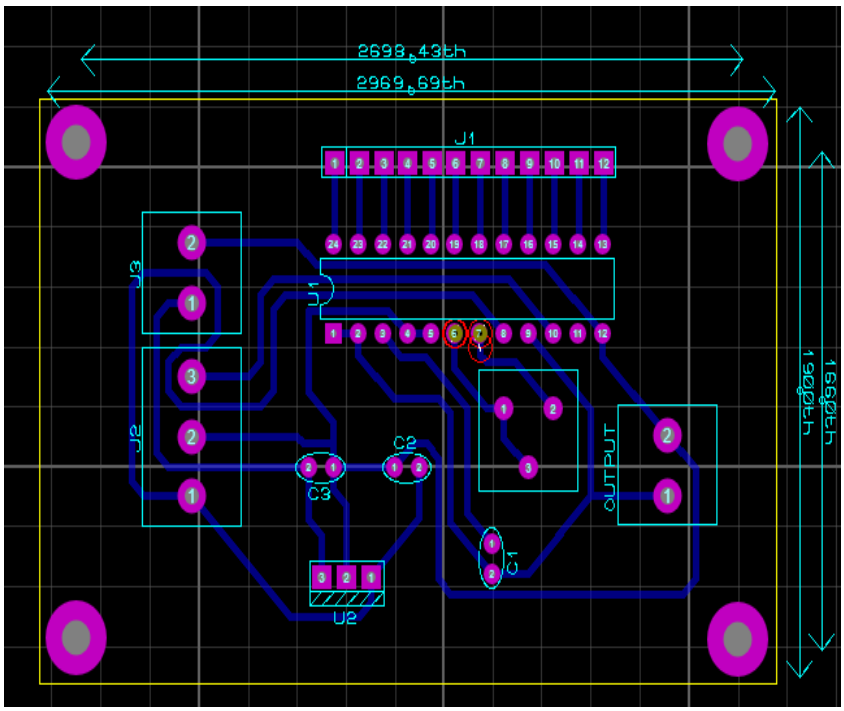


Figura 3.8 Esquemáticos del generador de señales

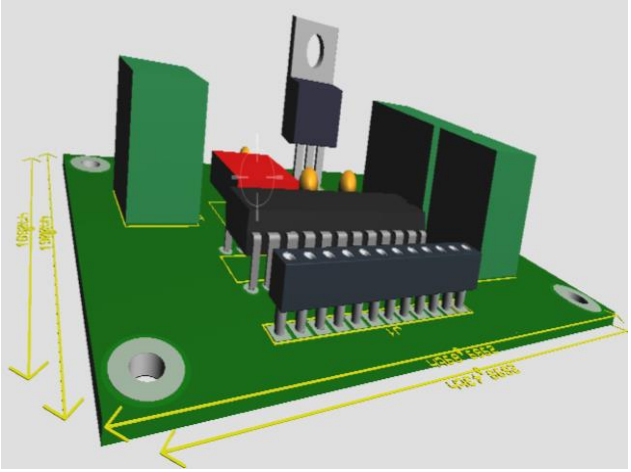


Figura 3.9 Circuito Generador de Señales (3D Visualizer)

Adicional al esquemático antes presentado del circuito generador realizamos una fuente bipolar de +12, -12 [V] para realizar la polarización del convertidor D/A AD767 y presentamos a continuación el esquemático del mismo

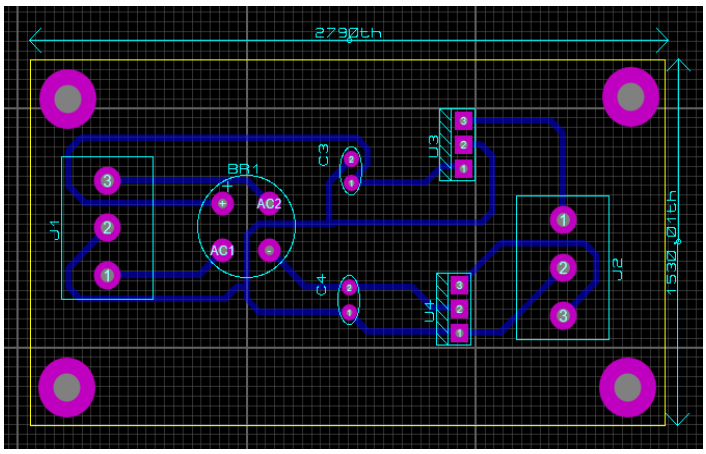


Figura 3.10 Esquemático del circuito Fuente Bipolar +12, -12 [v]

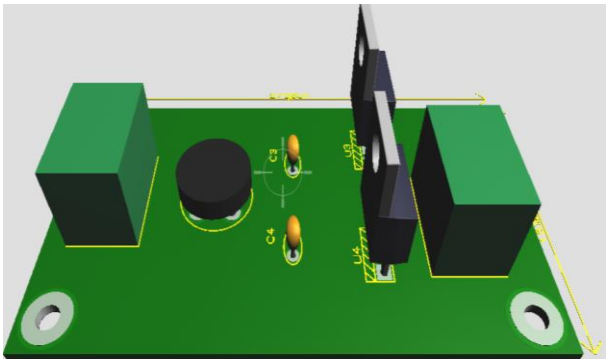


Figura 3.11 Circuito Fuente Bipolar +12, -12 [v] (3DVisualizer)

3.5 MÓDULOS DEL PROYECTO

El proyecto se encuentra organizado en 2 módulos presentados a continuación:

- **MÓDULO I:** El módulo del procesador embebido NIOS II.

  Este módulo se encarga de gestionar la comunicación entre el usuario con la interface gráfica y la obtención de datos de la onda que se desea generar, luego éste realiza el envío de la información ingresada por el usuario al módulo ***"Generador_Ondas".***

- **MÓDULO II:** Módulo generación de formas de onda por hardware denominado ***"Generador_Ondas"***.

El presente módulo construye la señal digital que se enviará al DAC y finalmente obtiene la onda análoga deseada.

3.6 COMPONENTES DEL SISTEMA

A continuación describimos los componentes que se utilizaron en el desarrollo del proyecto:

- Tarjeta Desarrollo DE0-Nano.
- Puerto de Expansión de la DE0-Nano JP1 (conexión de la pantalla).
- LCD TFT 4.3" (muestra la interfaz gráfica al usuario e interacción con el mismo).
- Puerto de Expansión de la DE0-Nano JP2 (conexión al convertidor D/A).

Convertidor D/A AD767 (Convierte la señal discreta a señal analógica, desde la salida del chip se realizan las mediciones con los equipos osciloscopio).

3.7 CREACIÓN DEL PROYECTO EN QUARTUS II

El proceso del diseño de hardware del sistema lo se desarrolló con la herramienta QUARTUS II.

Se inicia creando un proyecto nuevo siguiendo, ***File -> New Project Wizard.***

La siguiente Figura 3.2 muestra el proceso donde solicita la definición del directorio donde se va a trabajar, el nombre del proyecto y de la entidad de mayor jerarquía.

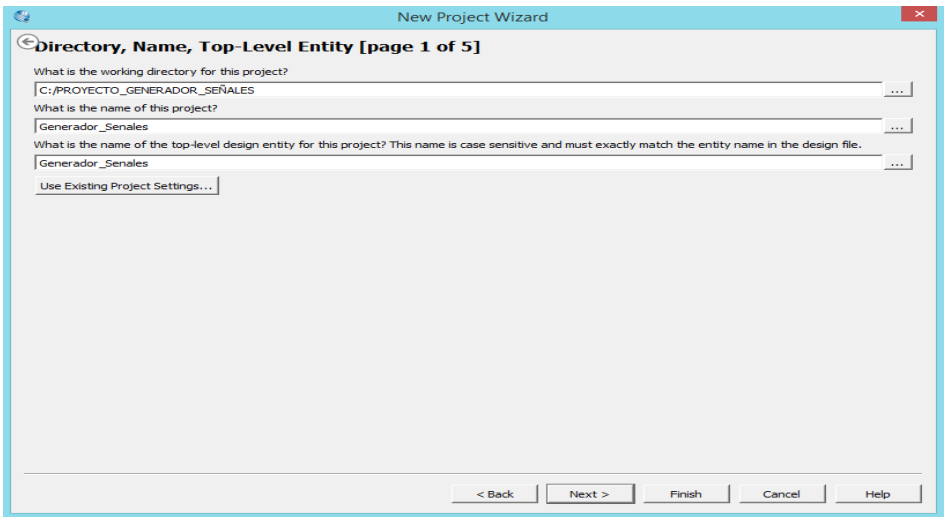


Figura 3.12 Pantalla de Definición de directorios y estructura del proyecto

La siguiente ventana permite ingresar archivos existentes es decir módulos .hdl o .vhdl al proyecto que sean de utilidad para la implementación del proyecto.

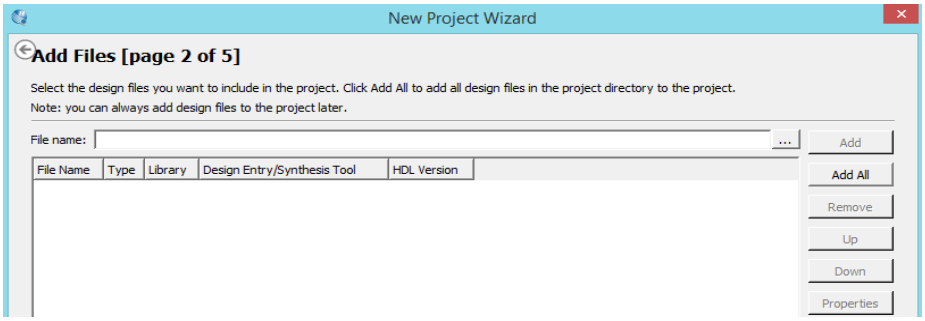


Figura 3.13 Pantalla de agregar, incluir archivos del proyecto

En el siguiente paso se define el dispositivo FPGA, el nombre y Familia. Para la tarjeta de desarrollo DE0-NANO, seleccionar:

- Familia: Cyclone IV E
- Nombre Dispositivo: EP4CE22F17C6

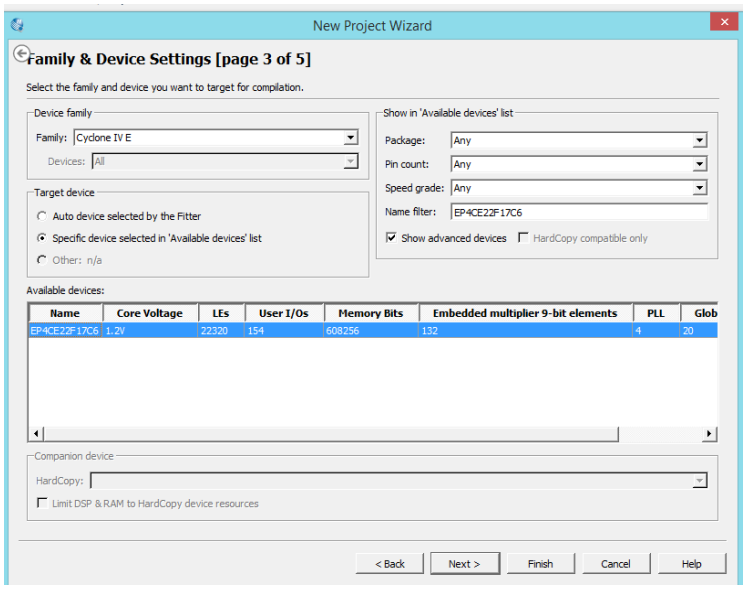


Figura 3.14 Pantalla de selección de la familia y dispositivo

En el cuarto paso permite especificar herramientas de otros fabricantes para la realización de diferentes etapas del proceso de diseño.

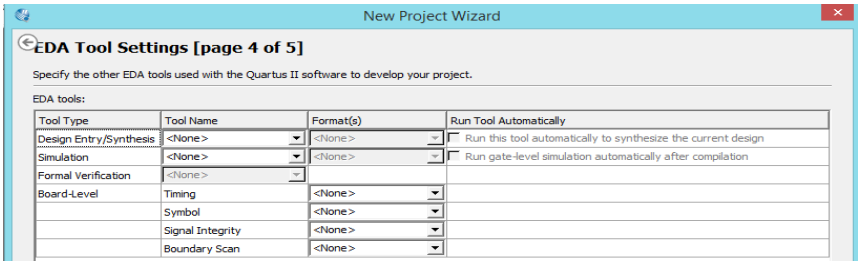


Figura 3.15 Pantalla de definición de herramientas de diseño de otros fabricantes a utilizar en el proyecto

En el último paso se puede apreciar el resumen de las características del proyecto:

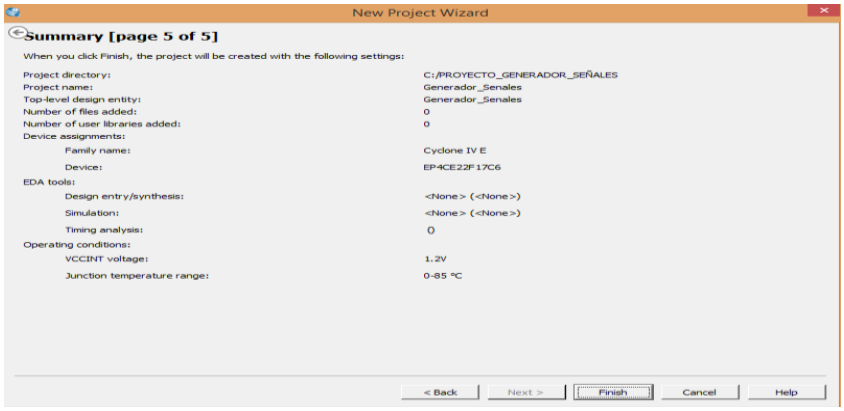


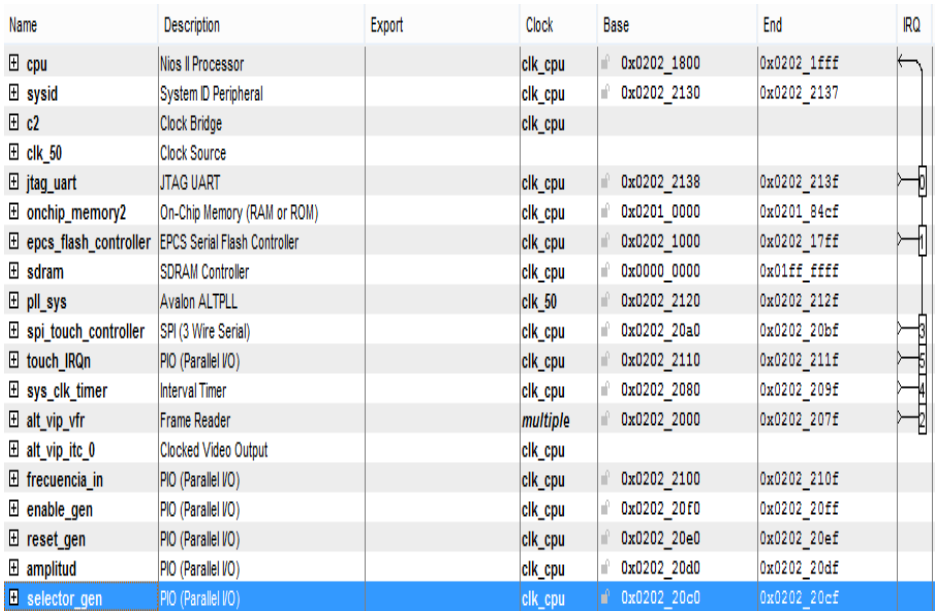
Figura 3.16 Pantalla de resumen de la creación del proyecto

Definida la estructura del proyecto, es posible proceder a realizar el diseño de la máquina central basado en el procesador NIOS II usando la herramienta QSYS.

3.7.1 COMPONENTES DE LA MÁQUINA (HARDWARE) ENQSYS

Una vez ya creado el proyecto en Quartus II se realiza el diseño del hardware con la ayuda de la herramienta QSYS, para esto seleccionar **Tools ->Qsys**

El Qsys posee módulos definidos configurables como: Procesador NIOS II, Memoria SDRAM, Memoria On-Chip, módulos PLL, módulo JTAG-UART, módulo Frame Reader, Clocked Video Output requeridos para utilizar la pantalla, son los que destacan y fueron utilizados para la creación del hardware del sistema. El diseño de hardware está formado por los siguientes componentes:



Name	Description	Export	Clock	Base	End	IRQ
cpu	Nios II Processor		clk_cpu	0x0202_1800	0x0202_1fff	
sysid	System ID Peripheral		clk_cpu	0x0202_2130	0x0202_2137	
c2	Clock Bridge		clk_cpu			
clk_50	Clock Source					
jtag_uart	JTAG UART		clk_cpu	0x0202_2138	0x0202_213f	0
onchip_memory2	On-Chip Memory (RAM or ROM)		clk_cpu	0x0201_0000	0x0201_84cf	
epcs_flash_controller	EPCS Serial Flash Controller		clk_cpu	0x0202_1000	0x0202_17ff	1
sdram	SDRAM Controller		clk_cpu	0x0000_0000	0x01ff_ffff	
pll_sys	Avalon ALTPLL		clk_50	0x0202_2120	0x0202_212f	
spi_touch_controller	SPI (3 Wire Serial)		clk_cpu	0x0202_20a0	0x0202_20bf	3
touch_IRQn	PIO (Parallel I/O)		clk_cpu	0x0202_2110	0x0202_211f	6
sys_clk_timer	Interval Timer		clk_cpu	0x0202_2080	0x0202_209f	4
alt_vip_vfr	Frame Reader		*multiple*	0x0202_2000	0x0202_207f	2
alt_vip_itc_0	Clocked Video Output		clk_cpu			
frecuencia_in	PIO (Parallel I/O)		clk_cpu	0x0202_2100	0x0202_210f	
enable_gen	PIO (Parallel I/O)		clk_cpu	0x0202_20f0	0x0202_20ff	
reset_gen	PIO (Parallel I/O)		clk_cpu	0x0202_20e0	0x0202_20ef	
amplitud	PIO (Parallel I/O)		clk_cpu	0x0202_20d0	0x0202_20df	
selector_gen	PIO (Parallel I/O)		clk_cpu	0x0202_20c0	0x0202_20cf	

Figura 3.17 Diseño de Hardware del sistema en QSYS

Procesador de NIOS II (CPU)

Es el módulo que se encarga de realizar los cálculos necesarios requeridos por el código C del sistema. EL tipo de

procesador que se escogió fue el NIOS II/f, la cual tiene todas las características para que el sistema sea rápido.

System ID Peripheral (sysid)

Identificador del sistema embebido. Es usado para la identificación dentro del NIOS II.

Clock Source (clk_50)

Fuente de flancos de reloj para el sistema embebido, este reloj funciona como señal de entrada para la PLL la cual es la que genera los diferentes tipos de relojes usados dentro del sistema.

JTAG UART

Módulo que se encarga de gestionar la comunicación entre el NIOS II SBT y el sistema embebido. Es usado para programar la tarjeta DE0-nano.

On-Chip Memory (onchip_memory2)

Se encarga de almacenar las direcciones de memoria donde se encuentra los diferentes componentes del procesador embebido NIOS II para luego ser referenciados dentro de la memoria RAM.

EPCS Serial Flash Controller

Controlador de la memoria flash de la DE0-nano la cual se usa para guardar el código de NIOS II.

SDRAM Controller (sdram)

Controlador de memoria sdram para la DE0-nano en la cual se guardan los datos de ejecución de NIOS II.

Avalon ALTPLL (pll_sys)

PLL la cual genera los diferentes relojes usados en el sistema.

SPI (3-wire serial) spi_touch_controller

Controlador de comunicación serial para la membrana touch.

PIO (touch_IRQn)

Pin de interface de comunicación para la señal IRQn de la pantalla táctil, la cual indica si la membrana táctil está siendo tocada o no.

Interval Timer (sys_clk_timer)

Timer para mantener un record del tiempo que lleva el Sistema ejecutándose.

Frame Reader (alt_vip_vfr)

Módulo que se encarga de leer de RAM las imágenes que se muestran en la pantalla.

Clocked Video Output (alt_vip_itc_0)

Módulo donde se parametriza las características de la pantalla donde se mostraran las imágenes guardadas en la RAM. Tales como resolución, formato de transmisión, cantidad de bits por pixel, etc.

PIO (pio_acumulador)

Bus de transmisión de datos desde el NIOS II hasta el módulo de hardware ***"Generador_Ondas"***, estos datos sirven para el cálculo de la frecuencia a generar.

PIO (pio_generar)

Pin que indica al módulo NCO empezar a generar la señal

PIO (pio_reset_gen)

Pin que indica al módulo NCO detener la señal que se está generando.

PIO (pio_selector)

Bus de datos que se envía al módulo NCO el cual indica el tipo de señal (seno, cuadrada, triangular, sierra) que se ha seleccionado.

PIO (pio_amplitud)

Bus de datos que se envía al módulo NCO con la información de la amplitud de la señal

3.7.2 IP CORES REQUERIDOS PARA EL SISTEMA

Para el diseño se utilizó el CORE WAVEFORM_GEN/NCO de la compañía de OpenCores.org. Este CORE viene con los archivos escritos en VHDL los cuales son:

- sincos_lut.vhd
- waveform_gen.vhd

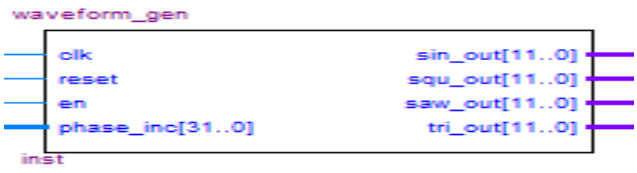


Figura 3.18 Bloque del CORE WAVEFORMGEN/NCO original

Se partió de la descripción de hardware del archivo waveform_gen.vhd y luego se personalizó según las necesidades de diseño. El Core fue descrito en el capítulo 2 en su versión original, a continuación se explica los cambios realizados en el CORE.

- Se agregó la señal de entrada “amplitud [6...0]”, se borró la salida “cos_out [11…0]” y se agregó la salida “tri_out [11…0]”.
- Se modificó el archivo (sincos_lut.vdh) reemplazando la tabla LUT del seno.
- Se agregó lógica digital en el archivo (waveform_gen.vhd) para generar la onda triangular. Se integró el IP CORE DIV para realizar modificación en la amplitud de la señales.

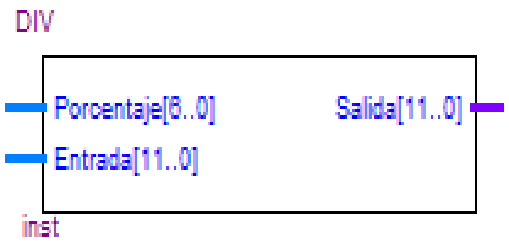


Figura 3.19 Bloque del IP CORE DIV

A continuación se explica el IP CORE DIV desarrollado en vhdl para manipular la amplitud de la señales, fue la estrategia a

desarrollar para realizar la variación de la amplitud instantánea según los parámetros enviados desde el interfaz gráfica

Este IP CORE es un divisor que recibe una entrada ***porcentaje [6…0]***, la representación de la señal denominada ***entrada [11…0]*** y la ***salida [11…0]*** señal variada en amplitud.

La finalidad de implementar este IP CORE fue básicamente de realizar la operación de porcentaje al valor ingresado para de esta manera obtener un voltaje de salida variable, nuestro convertidor D/A (AD767) con la configuración establecida con voltaje de salida de 0 a +10 [V], para esto realizamos un cálculo matemático, es decir el máximo valor aproximado de voltaje sería el de 10[V] el cual representaría nuestro 100%, los valores seleccionados desde la interfaz gráfica sobre la amplitud serían llevados a porcentaje función específica de este CORE para luego representarlo en voltaje.

Con esto se logra manejar la variación de voltaje netamente utilizando lógica digital ya que se enviará directamente al convertidor la señal ya variada en amplitud, sabiendo que es más sencillo y preciso manipular la señal digital que variar la señal analógica a la salida del D/A.

Previamente también se realizó un módulo de descripción de hardware llamado "***SELECTOR",*** que permite elegir el tipo de onda a generar.

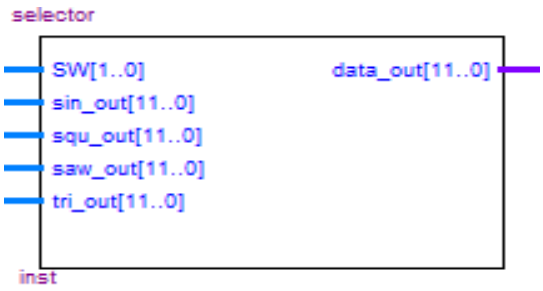


Figura 3.20 Bloque del IP CORE SELECTOR

Con todos estos CORES mencionados anteriormente se decidió agruparlos para obtener un solo módulo final con las funcionalidades del generador de señales y controlado por el procesador de NIOS II.

EL módulo final completo lo denominamos como **"Generador Ondas".**

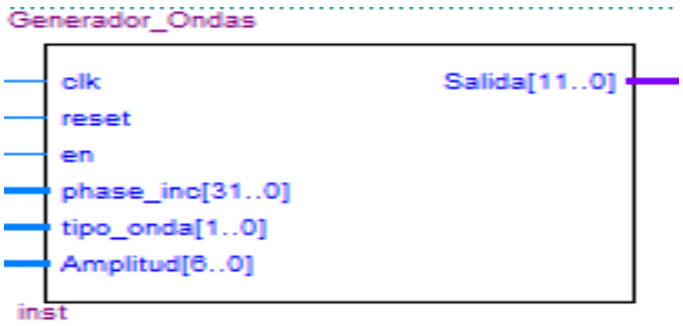


Figura 3.21 Bloque del IP CORE FINAL GENERADOR_ONDAS

3.7.3 AGREGAR EL IP CORE WAVEFORM_GEN/NCO BASADO EN LA IMPLEMENTACIÓN DE UN DDS A NUESTRO SISTEMA DE NIOS II

El proyecto tiene como función principal generar señales de tipo seno, cuadrada, triangular y rampa. Para lo cual se va a utilizar el CORE WAVEFORM_GEN/NCO basado en la tecnología DDS.

En primer lugar accedemos a la página **www.opencores.org,** donde en la sección de DSP se encontrará el CORE, luego se descarga y descomprime los archivos que están escritos en lenguaje VHDL, posteriormente se agrega los archivos al proyecto siguiendo los siguientes pasos:

1.-) Se selecciona el directorio principal del proyecto, ***Project ->Add/Remove Files on a Project.***

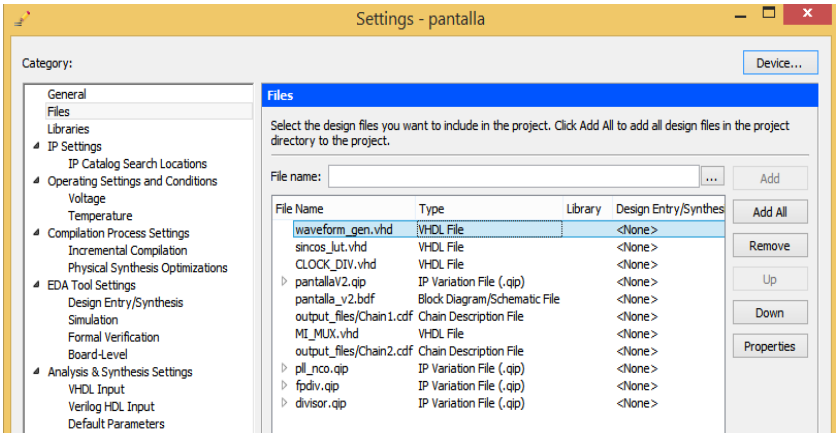


Figura 3.22 Agrega los archivos del CORE al proyecto principal

Luego de agregar el archivo .vhd del core a la ruta principal del proyecto es posible utilizarlo en el diseño de hardware.

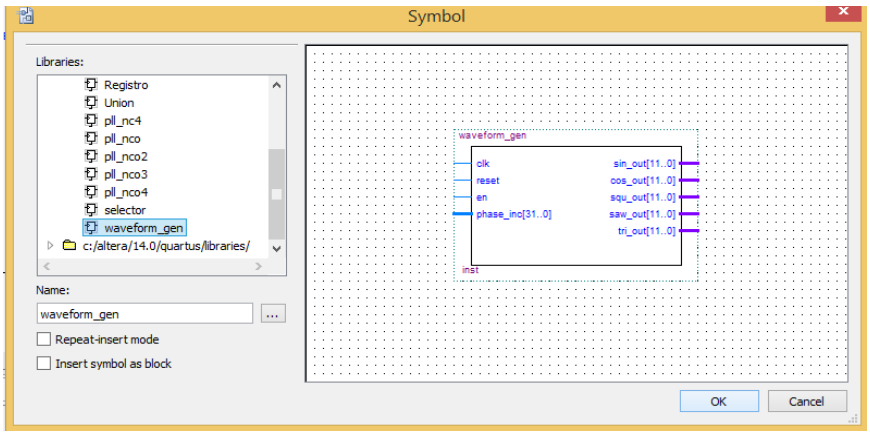


Figura 3.23 Bloque esquemático del CORE

3.7.4 MODIFICAR LA TABLA LUT DEL CORE WAVEFORM_GEN/NCO

En esta sección se describe el uso de la memoria ROM (tabla LUT), la misma que contiene los valores de amplitud de la onda seno, el CORE WAVEFORM/GEN posee una tabla LUT definida, estos valores han sido considerados con el bit de signo para obtener de esta manera el ciclo positivo y negativo de la onda, es decir para que estos valores respetando el signo sean enviados a un convertidor D/A, que dependiendo de su configuración convierta correctamente estos valores.

Para el desarrollo del proyecto se trabajó con el D/A AD767 en la configuración de **0 a +10 [V]** por lo que se debe modificar los valores de la tabla LUT para que el bit de signo no afecte a la interpretación que tiene el convertidor en base la configuración de hardware del DAC, es decir se desfasará los valores para de esta manera construir la onda correcta.

En las siguientes figuras se puede analizar con más detalle el efecto de modificar la LUT.

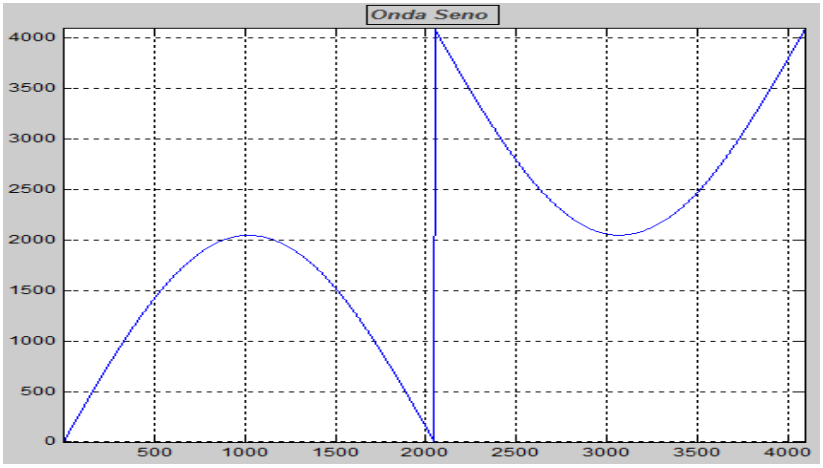


Figura 3.24 Gráfica de la onda seno con los valores de la LUT original

Como podemos ver en la Figura 3.24 los valores de la señal seno me grafican una onda de forma incorrecta para nuestra configuración del DAC.

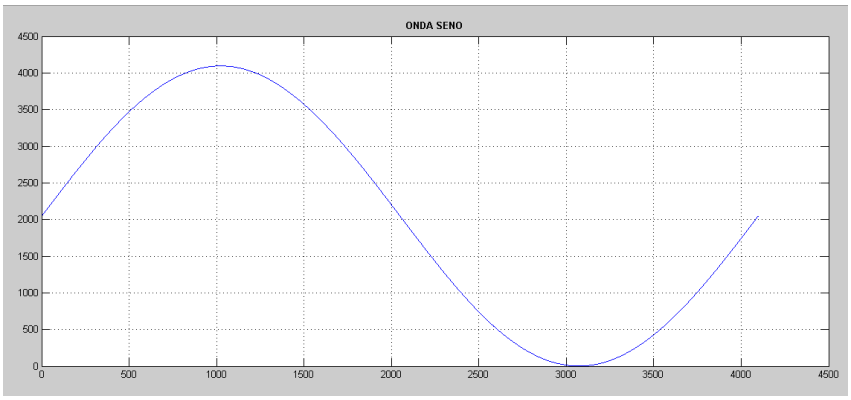


Figura 3.25 Gráfica de la onda seno con los valores de la LUT modificada

En la Figura 3.25 comprobamos que realizando el desfase entre los valores de la LUT obtenemos la señal senoidal correcta.

3.7.5 AGREGAR EL IP CORE “DIV”, AL SISTEMA PARA MANIPULAR LA AMPLITUD

Para agregar el IP CORE DIV (Divisor) al proyecto se realizó el siguiente procedimiento:

1.-) Se seleccionó el directorio principal del proyecto, ***Project ->Add/Remove Files on a Project.***

2.-) Se busca los archivos .vhd del core y luego se agregan al proyecto seleccionando el botón Add.

3.-) Después se verifica si el archivo del core se encuentra entre la lista de los archivos del proyecto y se aceptan.

4.-) Luego ya se tiene listo el core para utilizarlo en el diseño de hardware.

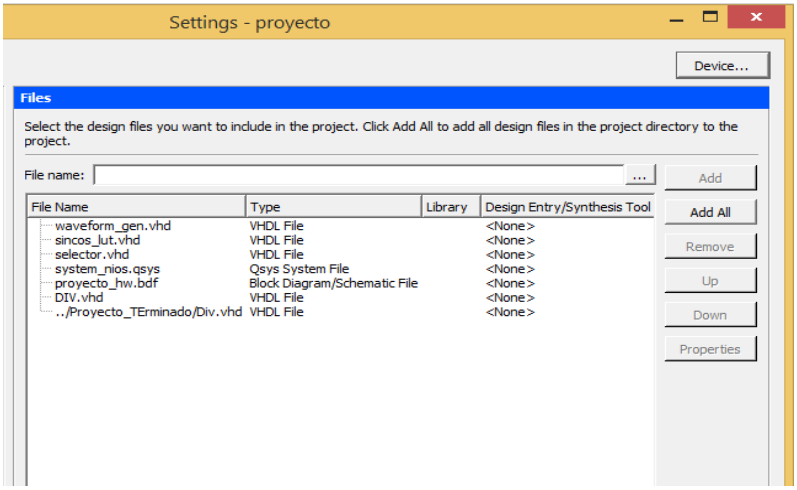


Figura 3.26 Ventana para agregar archivos externos al proyecto principal

3.7.6 COMPILACIÓN EN QUARTUS II DEL SISTEMA DISEÑADO EN QSYS

Una vez concluido el proceso de diseño del sistema hardware volvemos al entorno de trabajo de Quartus II, verificamos las conexiones del esquemático final.

Luego hacemos clic en: o vamos a las opción ***Processing ->StartCompilation***

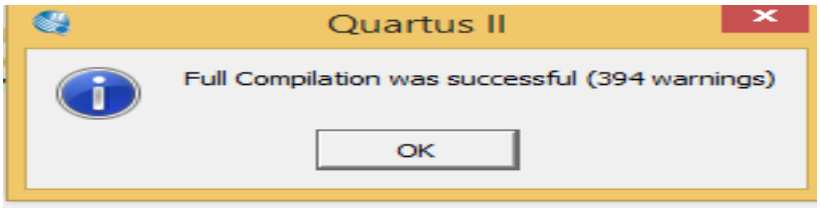


Figura 3.27 Ventana que indica compilación exitosa

Si la compilación es exitosa podemos revisar el diagrama RTL el cual me permite observar cada uno de los componentes del diseño de hardware final.

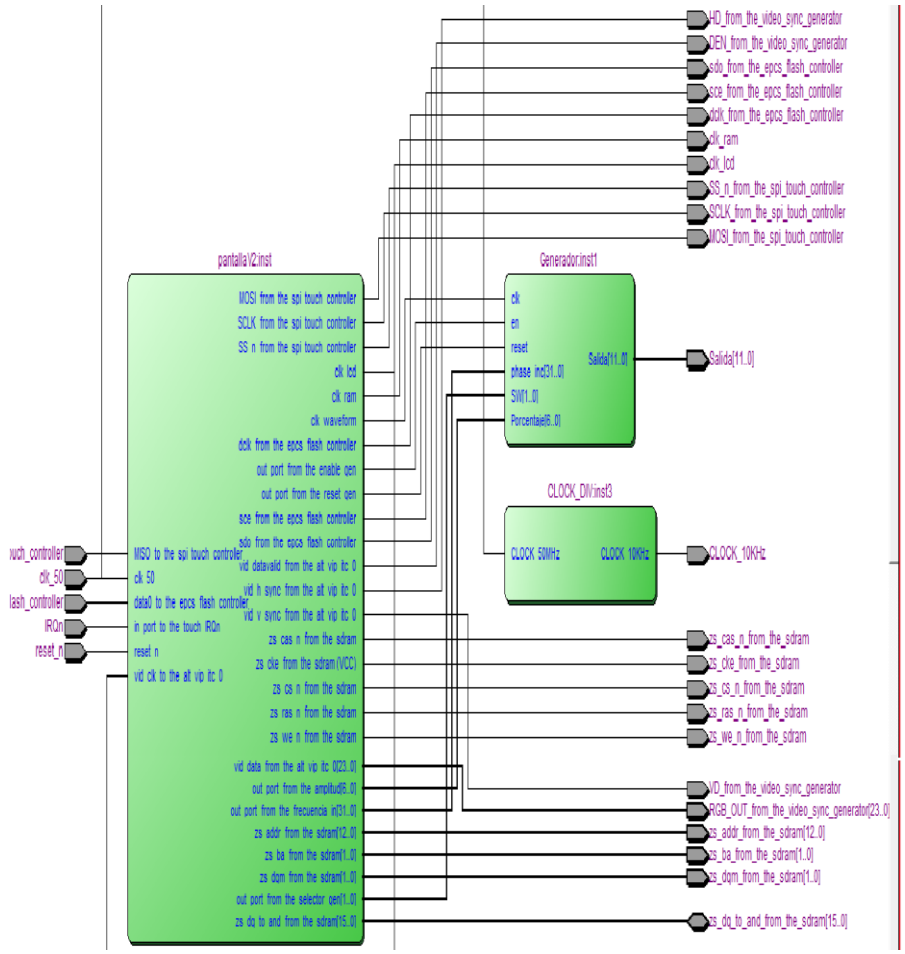


Figura 3.28 Diagrama RTL

3.7.7 ASIGNACIÓN DE PINES A UTILIZAR DE LA TARJETA DE0-NANO Y COMPILACIÓN

Realizamos el procedimiento de la asignación de pines, dando clic al ícono:  (Pin Planner)

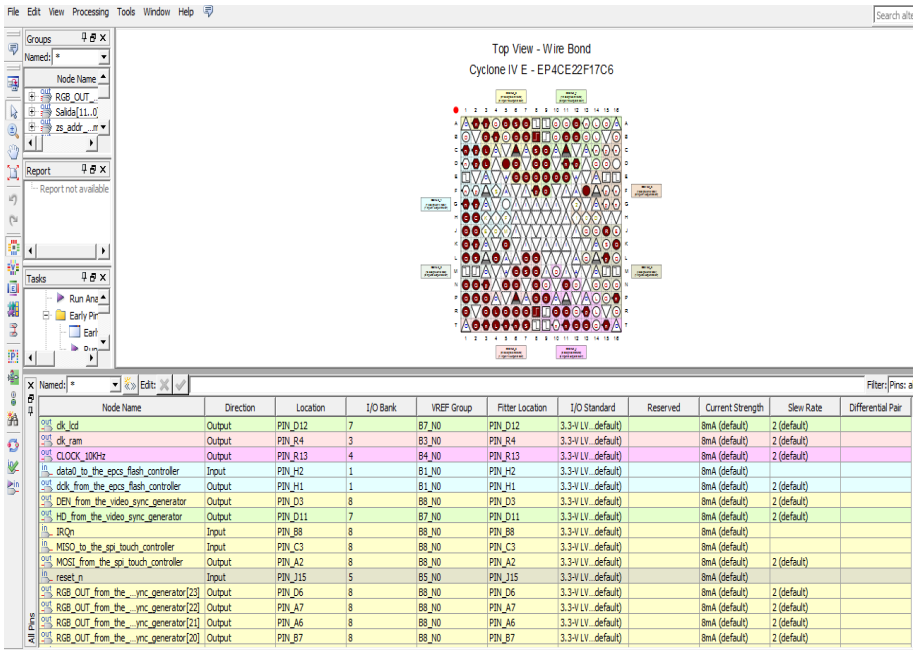


Figura 3.29 Ventana PIN PLANNER

Nos muestra la siguiente ventana donde debemos asignar el correspondiente pin a cada una de las terminales del diseño de hardware.

Para los componentes propios de la tarjeta de desarrollo como la memoria RAM, EPCS, podemos consultar la guía "DE0-Nano Manual de usuario 1.9".La asignación de pines realizada. En el proyecto la presentamos en la siguiente tabla:

Tabla 5 Asignación de pines principales del proyecto

SEÑAL	***DIRECCIÓN***	***PINES***	***DESCRIPCIÓN***
Salida[0]	Output	PIN_F13	Señales de salida hacia el convertidor DAC, representan los bits de la forma de onda
Salida[1]	Output	PIN_T15	
Salida[2]	Output	PIN_T13	
Salida[3]	Output	PIN_T12	
Salida[4]	Output	PIN_T11	
Salida[5]	Output	PIN_R11	
Salida[6]	Output	PIN_R10	
Salida[7]	Output	PIN_P9	
Salida[8]	Output	PIN_N11	
Salida[9]	Output	PIN_K16	
Salida[10]	Output	PIN_L15	
Salida[11]	Output	PIN_P16	
clk_ram	Output	PIN_R4	Señal de Reloj para la RAM (100Mhz)
clk_lcd	Output	PIN_R5	Señal de Reloj para la pantalla (9Mhz)
clk_50	Input	PIN_R8	Señal de Reloj (50Mhz)
clk_10Khz	Output	PIN_R13	Señal de Reloj (10Khz), hacia el blackligthde la pantalla
IRQn	Input	PIN_B8	Señal de entrada, que es leída por el procesador para detectar los eventos touch
DEN_Video	Output	PIN_D3	Señal de salida para habilitar la pantalla
MISO_Spi	Input	PIN_C3	Señal de entrada para la comunicación SPI de la pantalla
MOSI_Spi	Output	PIN_A2	Señal de salida para la comunicación SPI de la pantalla
SCLK_Spi	Output	PIN_A3	Señal de salida SCK para la comunicación SPI de la pantalla
SSN_Spi	Output	PIN_A3	Señal de salida para el chip select SPI de la pantalla
VD_VIDEO	Output	PIN_B12	Señal de sincronización vertical para la pantalla
HD_VIDEO	Output	PIN_D11	Señal de sincronización horizontal para la pantalla

Tabla 6 Asignación de pines para los bits de colores de la pantalla

SEÑAL	*DIRECCIÓN*	*PINES*	*DESCRIPCIÓN*
RGB_out[0]	Output	PIN_C9	Señales de la salida de bits que representan los colores de la pantalla
RGB_out[1]	Output	PIN_E9	
RGB_out[2]	Output	PIN_E11	
RGB_out[3]	Output	PIN_D9	
RGB_out[4]	Output	PIN_C11	
RGB_out[5]	Output	PIN_E10	
RGB_out[6]	Output	PIN_A12	
RGB_out[7]	Output	PIN_B11	
RGB_out[8]	Output	PIN_C8	
RGB_out[9]	Output	PIN_D8	
RGB_out[10]	Output	PIN_F9	
RGB_out[11]	Output	PIN_F8	
RGB_out[12]	Output	PIN_A5	
RGB_out[13]	Output	PIN_B4	
RGB_out[14]	Output	PIN_B6	
RGB_out[15]	Output	PIN_D5	
RGB_out[16]	Output	PIN_A5	
RGB_out[17]	Output	PIN_B4	
RGB_out[18]	Output	PIN_B6	
RGB_out[19]	Output	PIN_D5	
RGB_out[20]	Output	PIN_B7	
RGB_out[21]	Output	PIN_A6	
RGB_out[22]	Output	PIN_A7	
RGB_out[23]	Output	PIN_D6	

3.7.8 PROGRAMAR EL DISEÑO DEL SISTEMA EN LA FPGA

Luego de haber compilado con éxito el diseño de hardware, son generados los archivos requeridos para programar la FPGA. Para esto realizamos el siguiente procedimiento:

- Ir a ***Tools->Programmer*** 
- Se muestra la ventana de configuración del dispositivo
- Verificamos que la tarjeta de desarrollo esté alimentada y conectada a la PC mediante el USB (habilitado el driver USB-Blaster) y en modo JTAG.
- Se procede a seleccionar el archivo de descripción de hardware .sof
- Luego presionamos el botón START y en el extremo superior vemos la barra de progreso que alcanza un 100%, esto indica que la programación de la FPGA ha sido exitosa.

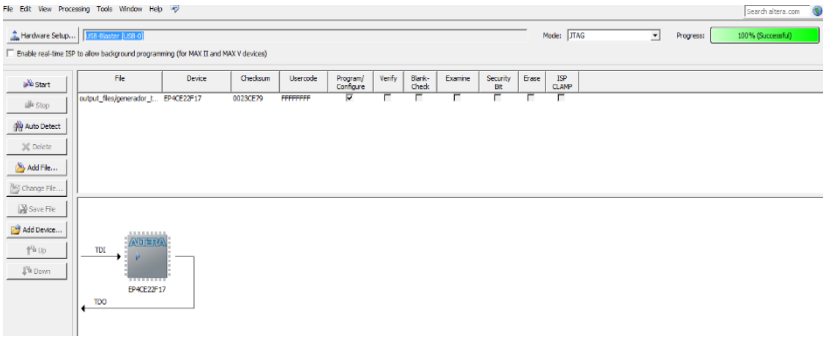


Figura 3.30 Venta para realizar la programación del Hardware en la FPGA

Una vez programada la FPGA tenemos listo el hardware del sistema para ser utilizado en las aplicaciones y funcionalidades que se implementarán mediante la programación en lenguaje C, es decir ya se puede realizar el software del sistema.

3.8 OBTENCIÓN E IDENTIFICACIÓN DE DATOS

Para la obtención de los datos en el proyecto utilizamos una pantalla Táctil que a través de una Interfaz Gráfica GUI permite realizar las diversas instrucciones requeridas para el sistema, está pantalla es un dispositivo que mediante un contacto directo sobre su superficie permite

la entrada de datos**,** posee una resolución de 480x272, con comunicación SPI.

Actualmente las pantallas táctiles se han difundido en su mayoría desde el desarrollo tecnológico y aparición de múltiples dispositivos móviles, existen varias tecnologías de las mismas, entre ellas las pantallas táctiles resistiva utilizada en nuestro proyecto, que consisten en una membrana de vidrio con una delgada capa resistiva y una capa conductiva. La capa resistiva tiene una lámina protectora y entre ambas capas existen pequeños puntos aislantes que las separan.

Cuando la pantalla es tocada, las capas hacen contacto con el vidrio, generando un voltaje eléctrico, indicando así la coordenada del punto tocado.

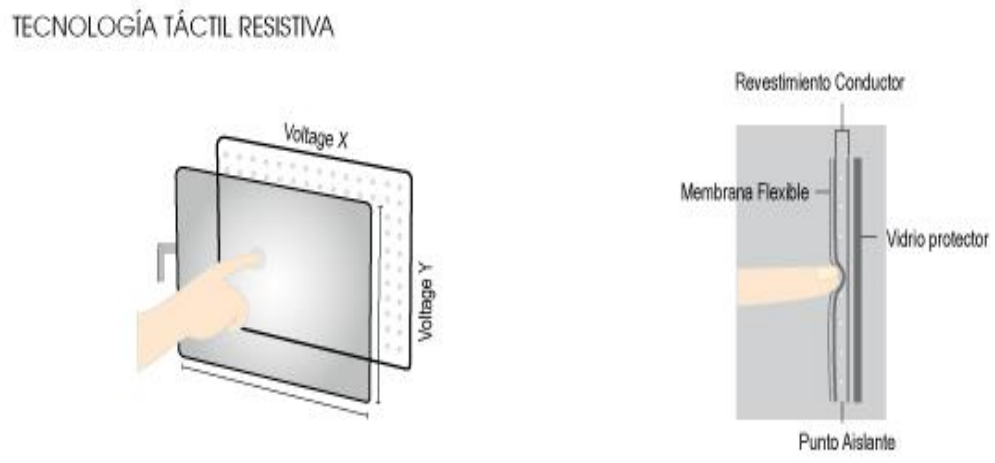


Figura 3.31 Pantalla Táctil Resistiva [22]

3.9 IMPLEMENTACIÓN DEL PROGRAMA EN NIOS II SBT

Para la implementación del software del sistema abrimos la aplicación NIOS II SBT y crear un nuevo proyecto.

- ***File -> New ->Nios II Aplicaction and BSP from Template***

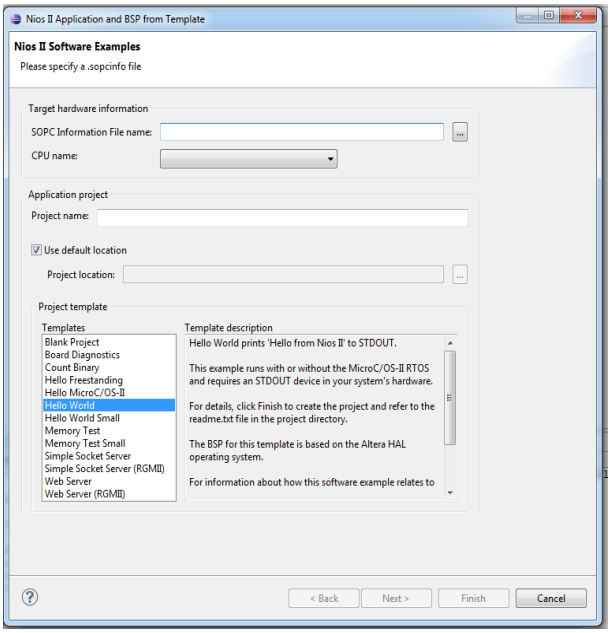


Figura 3.32 Ventana de creación del proyecto en NIOS II SBT

Luego se agrega el archivo con la información de hardware del procesador embebido NIOS II, el cual tiene extensión .socpinfo. Este

archivo es creado automáticamente cuando se genera el hardware del procesador en Qsys.

3.9.1 UTILIZACIÓN DEL SISTEMA OPERATIVO uC/OS II PARA TRABAJAR MEDIANTE MULTITASK

Para el uso del sistema operativo uC/OS II, al momento de elegir una plantilla del proyecto se selecciona “HelloMicroC/OSII”.

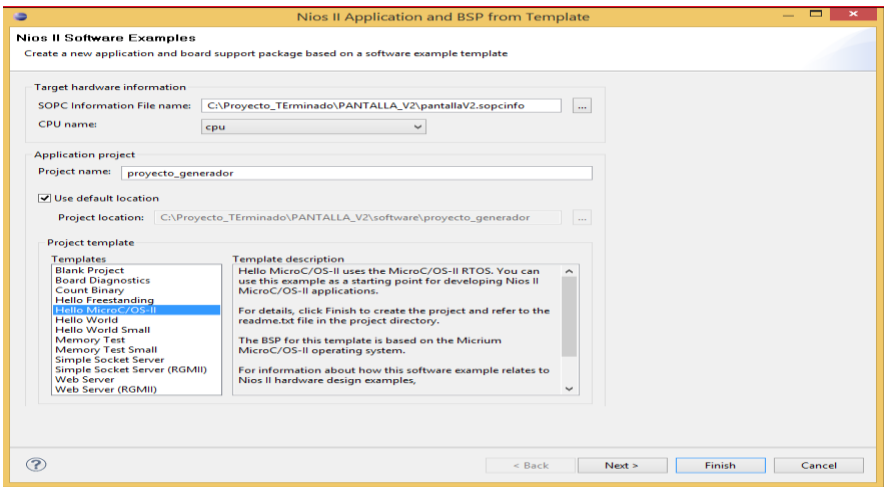


Figura 3.33 Ventana para crear el proyecto en NIOS II SBT

Al seleccionar esta plantilla, automáticamente se agregan las librerías necesarias para poder usar el sistema operativo.

```
#include <stdio.h>
#include "includes.h"
/* Definition of Task Stacks */
#define   TASK_STACKSIZE       2048
OS_STK    task1_stk [TASK_STACKSIZE];
OS_STK    task2_stk [TASK_STACKSIZE];
/* Definition of Task Priorities */
#define TASK1_PRIORITY      1
#define TASK2_PRIORITY      2
void task1 (void* pdata) {
       while (1){
              printf ("Hello from task1\n");
              OSTimeDlyHMSM (0, 0, 3, 0);
       }
}
void task2 (void* pdata) {
       while (1) {
              printf ("Hello from task2\n");
              OSTimeDlyHMSM (0, 0, 3, 0);
       }
}
int main (void) {

OSTaskCreateExt (task1,
NULL,
(void *)&task1_stk [TASK_STACKSIZE-1],
TASK1_PRIORITY,
TASK1_PRIORITY,
task1_stk,
TASK_STACKSIZE,
NULL,
0);

OSTaskCreateExt (task2,
NULL,
(void *)&task2_stk [TASK_STACKSIZE-1],
TASK2_PRIORITY,
TASK2_PRIORITY,
task2_stk,
TASK_STACKSIZE,
NULL,
0);
OSStart ();
return 0;
}
```

Figura 3.34 Código de la plantilla inicial del "Hello uC/OS II"

3.9.2 PROGRAMACIÓN EN LENGUAJE C PARA REALIZAR LAS FUNCIONALIDADES DEL GENERADOR DE SEÑALES EN LA PANTALLA

En esta sección describimos el software en detalle. La organización del proyecto quedo definido de la siguiente manera:

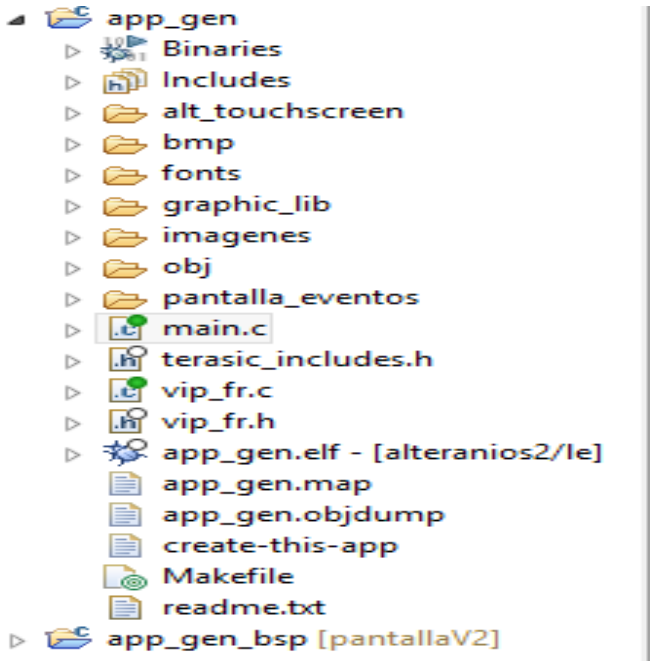


Figura 3.35 Esquema del proyecto en NIOS II SBT

Las funciones de interacción y dibujo del diseño de la interfaz gráfica, implementadas para nuestra estrategia de desarrollo se encuentran en la carpeta ***"pantalla_eventos",*** aquí

tenemos el procedimiento "***dibuja_gui",*** descrita anteriormente en la Figura 3.3, que se encarga de dibujar la interfaz gráfica en la pantalla, toma las imágenes cargadas en código C y las coloca en diferentes coordenadas, también dibuja cuadros y texto completando así el GUI. El procedimiento ***"mostrar_valores",*** presentado en la Figura 3.4, muestra los valores de amplitud y frecuencia. También se tiene en esta carpeta el procedimiento ***"cambios pantalla"***

```
Void cambios_pantalla (touch_control *pic_viewer,
VIP_FRAME_READER *pReader) {
if(touch->ui_command==UI_MENOS_AMP)
      DispIMAGE((void*)display, 40, 47,&BTN_ATRAS_2);
else if (touch->ui_command==UI_MAS_AMP)
      DispIMAGE((void*)display, 170,
47,&BTN_ADELANTE_2);

else if (touch->ui_command==UI_MENOS_FREC)
      DispIMAGE((void*)display, 205,
47,&BTN_ATRAS_2);

else if (touch->ui_command==UI_MAS_FREC)
      DispIMAGE((void*)display, 332,
47,&BTN_ADELANTE_2);
else if (touch->ui_command==0){
      DispIMAGE((void*)display, 170,
47,&BTN_ADELANTE);
DispIMAGE((void*)display, 40, 47,&BTN_ATRAS);
      DispIMAGE((void*)display, 332,
47,&BTN_ADELANTE);
}
```

Figura 3.36 Fragmento del código para realizar el cambio de botones al presionar la pantalla

3.10 INTERFAZ DE USUARIO A MOSTRAR EN LA PANTALLA TÁCTIL LCD TOUCH

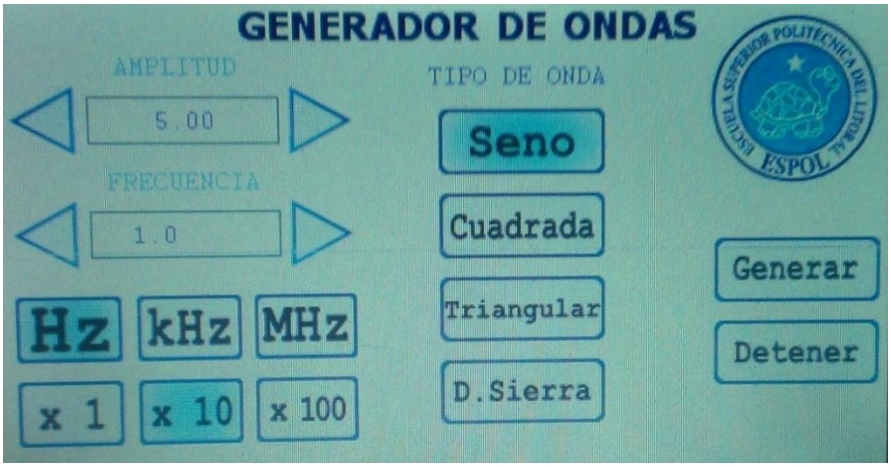


Figura 3.37 Interfaz Gráfica Inicial

Inicialmente realizamos el diseño de la interfaz gráfica mostrada en la Figura 3.37, pero por motivos de espacio y para poder brindar una mayor interactividad a los usuarios del equipo realizamos un diseño final con una mejor organización de los componentes de la GUI, botones, imágenes y más intuitiva para los usuarios.

Figura 3.38 Interfaz Gráfica Final del Generador de Señales Básicas

CAPÍTULO 4

4. PRUEBAS Y ANÁLISIS DE RESULTADOS

En esta sección se realizan las pruebas del sistema con sus respectivos análisis de resultados.

4.1 ESCENARIO A: PRUEBA DEL SISTEMA NIOS II PARA CARGAR LA INTERFAZ EN LA PANTALLA

En esta prueba vamos a utilizar las librerías gráficas de altera, los drivers para trabajar en conjunto al Hardware ya establecido.

Previo a esto debemos revisar que las señales principales de la LCD que estén conectadas correctamente y controlar la señal blackligth con un PWM para que no consuma mucha corriente, ya que si no se lo utiliza afectaría directamente el desempeño de la pantalla.

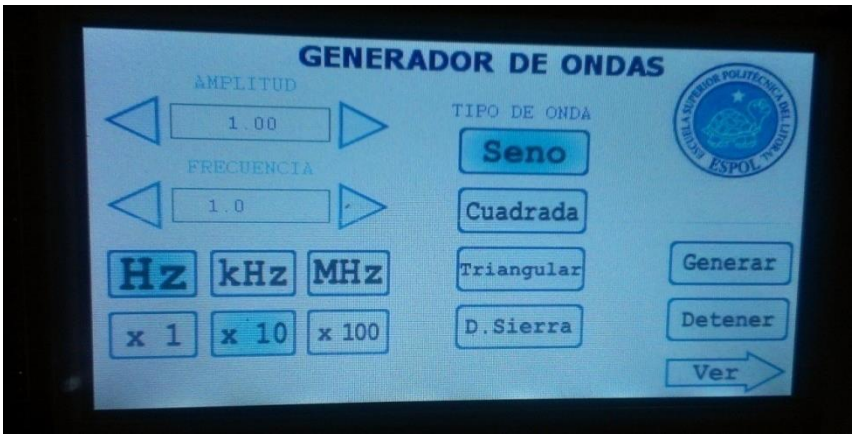


Figura 4.1 Interfaz gráfica de pruebas

Como podemos ver en la Figura 4.1 tenemos los botones que permitirán al usuario interactuar con la GUI y a su vez seleccionar los parámetros de las señales.

4.2 ESCENARIO B: PRUEBA DEL CORE WAVEFORMGEN/NCO

En la siguiente prueba realizamos un diseño de hardware donde probamos las funcionalidades del Core encargado de realizar el proceso de generar las formas de onda, revisamos sus limitantes y analizamos la mejor manera de adaptarlo a nuestras necesidades de desarrollo.

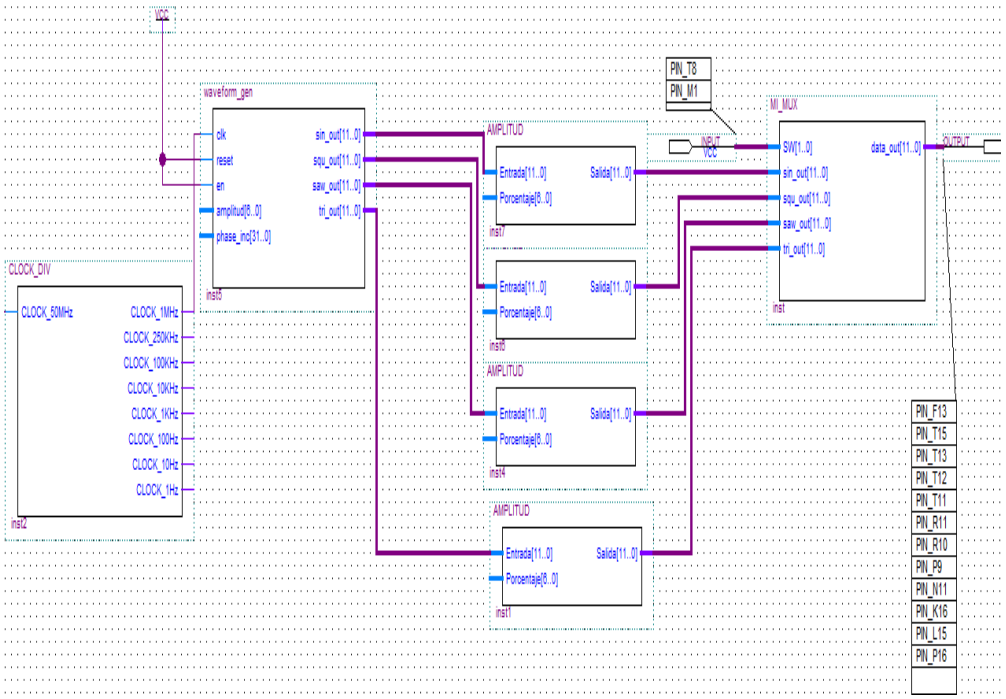


Figura 4.2 Esquemático prueba CORE WAVEFORMGEN/NCO

4.3 ESCENARIO C: PRUEBA DEL HARDWARE CON EL CORE Y EL CONVERTIDOR D/A PARA LA LECTURA DE LAS SEÑALES

En este escenario configuramos los puertos de salida PIO para obtener los 12 bits que construyen las señales, con el diseño de hardware del escenario B se realizó esta prueba en conjunto a la configuración del convertidor D/A AD767 de 0 +10 [v], para obtener las lecturas de las formas de onda mediante un osciloscopio.

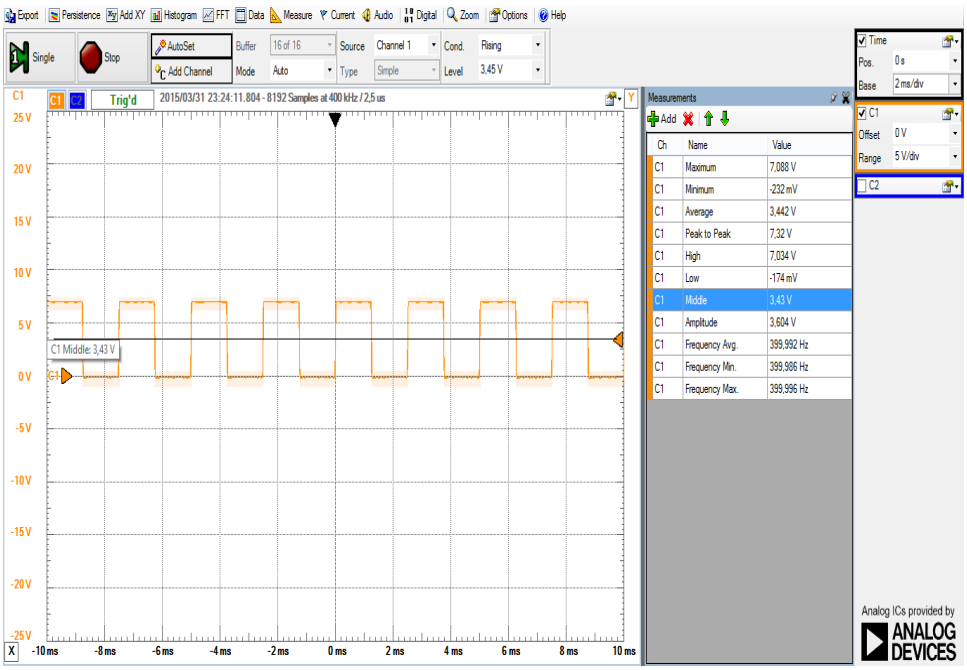


Figura 4.3 Lectura de la forma de onda cuadrada obtenida en la prueba

4.4 ESCENARIO D: PRUEBA DELATABLA LUT DEL CORE WAVEFORMGEN/NCO MODIFICADA

Al realizar las pruebas y como lo describimos anteriormente el core waveform_gen posee una tabla LUT donde se encuentran los valores de la señal seno y coseno pero al realizar las lecturas previas de las señales medidas con el osciloscopio se obtiene una forma de onda incorrecta, esto se da debido a que los valores de la tabla LUT no estaban de acuerdo a la configuración del convertidor D/A, en nuestro caso con la configuración de 0 a +10V.

Por lo que tuvimos que modificar los valores de la tabla LUT para obtener la señal correcta.

A continuación mostramos en la tabla los primeros 20 valores de la LUT original y los valores modificados, la tabla posee una resolución de 12 bits por lo tanto tendrá 4096 valores.

Tabla 7 Tabla comparativa de los valores de la LUT

# Datos	Valor Decimal Original	Valor Hexadecimal Original	Valor Decimal Modificado	Valore Hexadecimal Modificado	GRADOS
1	0	000	2048	800	0,088
2	3	003	2051	803	0,176
3	6	006	2054	806	0,264
4	9	009	2057	809	0,352
5	13	00d	2061	80D	0,44

6	16	010	2064	810	0,528
7	19	013	2067	813	0,616
8	22	016	2070	816	0,704
9	25	019	2073	819	0,792
10	28	01c	2076	81C	0,88
11	31	01f	2079	81F	0,968
12	35	023	2083	823	1,056
13	38	026	2086	826	1,144
14	41	029	2089	829	1,232
15	44	02c	2092	82C	1,32
16	47	02f	2095	82F	1,408
17	50	032	2098	832	1,496
18	53	035	2101	835	1,584
19	57	039	2105	839	1,672
20	60	03c	2108	83C	1,76
...	...	...	...	...	...

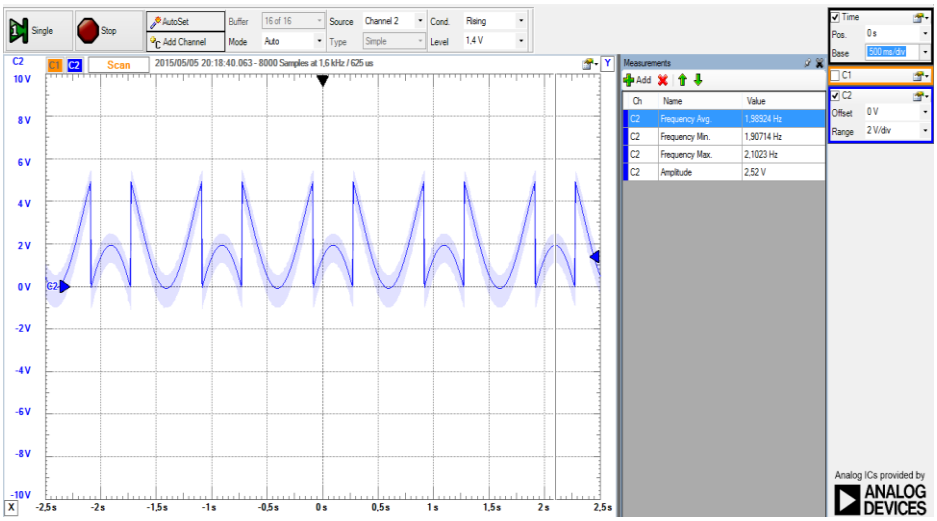


Figura 4.4 Lectura de la onda seno obtenida en la prueba con la LUT original

Realizando el cambio en los valores de la LUT obtenemos la señal mostrada a continuación:

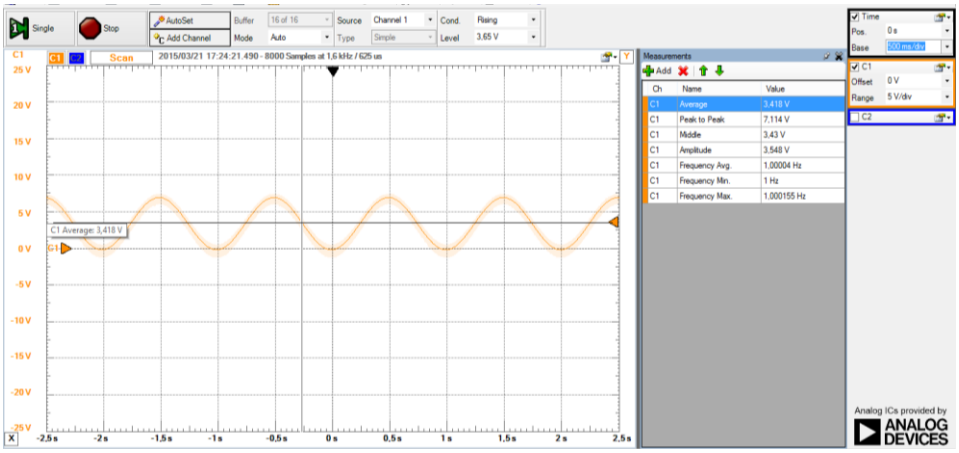


Figura 4.5 Lectura de la forma de onda obtenida en la prueba con la LUT modificada

4.5 ESCENARIO E: PRUEBAS DE INTERACCIÓN

En el siguiente escenario realizamos las pruebas del proyecto completo la integración de las dos fases del sistema, la pantalla y el generador de señales.

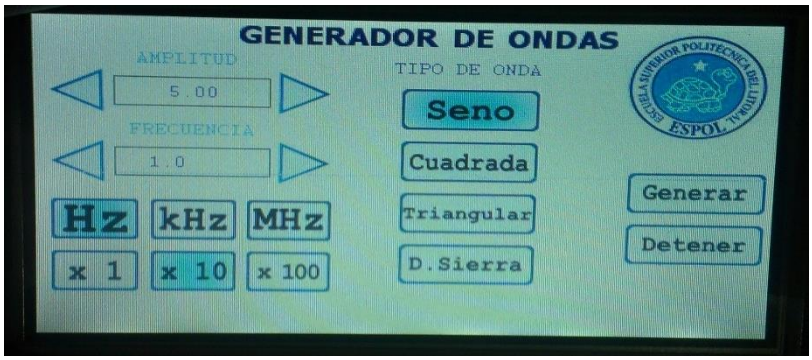


Figura 4.6 Prueba Interacción con la GUI

```
Bandera Event: 3 IRQN: 0
coordenada x=318 y=78
Bandera Event: 0 IRQN: 0
coordenada x=282 y=79
Onda Seno detectado
```

Figura 4.7 Prueba que imprime por consola las coordenadas al realizar el evento táctil

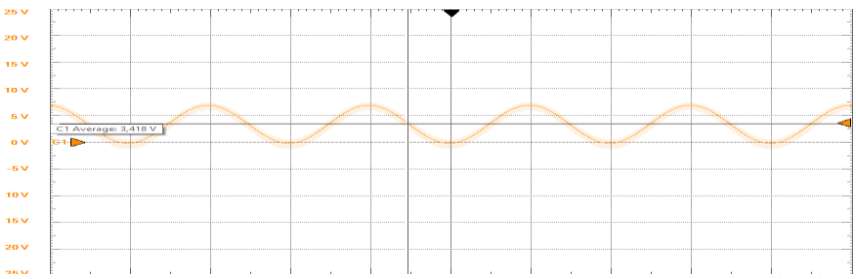


Figura 4.8 Señal generada de prueba onda seno

4.6 ESCENARIO F: COMPARACIÓN DEL PROYECTO CON EL SEGMENTO DEL MERCADO

Para realizar el siguiente estudio realizamos encuestas a 50 estudiantes de la ESPOL de la facultad FIEC, se les hizo probar el generador de funciones digital en el laboratorio de electrónica. Luego de probar el generador de funciones, se les hizo una encuesta con las siguientes preguntas:

1.-) ¿Le agrada la presentación física del generador de funciones en comparación con un generador análogo? SI/NO.

2.-) ¿La interfaz gráfica es entendible y amigable? SI/NO.

3.-) ¿El rango de frecuencias en el que funciona el generador es adecuado para un laboratorio de electrónica? SI/NO

4.-) ¿Las funcionalidades del generador de funciones es suficiente para la mayoría de aplicaciones? SI/NO

5.-) ¿Si este generador fuera un producto comercial lo compraría? SI/NO

6.-) ¿Si este generador de funciones también tuviese la funcionalidad de un osciloscopio lo compraría? SI/NO

Los resultados de la encuesta fueron:

Tabla 8 Tabla de resultados de las encuestas

PREGUNTA	SI	NO
Le agrada la presentación física del generador de funciones en comparación con un generador análogo?	42	8
La interfaz gráfica es entendible y amigable?	40	10
El rango de frecuencias en el que funciona el generador es adecuado para un laboratorio de electrónica?	36	14
El rango de voltaje es adecuado para pruebas en un laboratorio de electrónica?	37	13
Las funcionalidades del generador de funciones es suficiente para la mayoría de sus aplicaciones?	39	11
Si este generador fuera un producto comercial, lo compraría?	30	20
Si este generador de funciones también tuviese la funcionalidad de un osciloscopio lo compraría?	48	2

Los resultados de la encuesta nos muestran que el generador de funciones en general cumple con las expectativas de los usuarios. Se desea que tuviese un mayor espectro de funcionamiento en términos de frecuencia y algo en voltaje, lo cual se puede solucionar eligiendo un convertidor DAC de mayores características de funcionamiento, adicionalmente la tarjeta DE0-nano puede generar la señal digital hasta

150 MHZ lo cual se podría usar un convertidor DAC que genere ondas de hasta 15 MHz.

Algo que les gustaría a los usuarios adicionalmente, es que también pueda funcionar como un osciloscopio, lo cual es posible usando el ADC que se encuentra incluido dentro de la tarjeta DE0-NANO.

CONCLUSIONES

1. En base a las pruebas realizadas en el presente trabajo podemos llegar a la conclusión de que las principales limitantes para un equipo generador de señales digital es el convertidor D/A, debido a que la señal análoga depende la velocidad de conversión del DAC el cual determina el rango de frecuencia de un generador de señales.

2. El Core ***"waveformgen/NCO"*** nos ayuda a realizar el diseño de hardware modular y al utilizar este módulo para que genere la onda digital por hardware, podemos optimizar los recursos de la

FPGA y del procesador de NIOS II.

3. La técnica DDS es la mejor estrategia para generar formas de onda, ya que construye señales más precisas con menor rango de distorsión.

4. En base a las pruebas podemos definir que para utilizar la técnica DDS debemos asegurarnos que los valores de la tabla LUT sean acordes al convertidor D/A que se vaya a utilizar. Además para tener una mejor resolución de onda es mejor usar una tabla LUT más grande.

5. El filtro paso bajo aplicado a la señal análoga que da el DAC, elimina las ondas armónicas de alta frecuencia de la señal generada, dando una mejor resolución, con menor ruido.

6. Las características de impedancia y corriente máxima del generador de señales están atadas a las características técnicas del DAC que se usará en la implementación del proyecto.

RECOMENDACIONES

1. Es recomendable realizar la investigación y lectura previa de las hojas de datos de los componentes a utilizar en un generador de señales, así como también leer el manual de usuario de la tarjeta de desarrollo.

2. Leer toda la hoja de datos de la LCD Touch a utilizar para determinar la interfaz de comunicación, resolución de la pantalla, bits de colores y asignación de pines.

3. Al trabajar con la pantalla LCD Touch se debe manipular la señal Blackligtht con un PWM o una resistencia para bajar el consumo de corriente de la tarjeta de Desarrollo, caso contrario la comunicación entre la DE0- Nano y la membrana Touch, fallará.

4. Separar en tareas la funcionalidad del proyecto a realizar de tal forma que al momento de realizar la implementación usando el sistema operativo “uC/OS II” el proyecto se encuentre lo suficientemente modularizado, para reducir la complejidad de la programación, código más escalable y reducir el tiempo de resolución de problemas.

5. Para el desarrollo de un equipo generador de señales se recomienda tener a la mano un osciloscopio.

BIBLIOGRAFÍA

[1] Enciclopedia libre, System-on-chip, http://es.wikipedia.org/wiki/System_on_a_chip, fecha de consulta julio 2014.

[2] P.Buckley, PSoC power monitoring and fan control solutions aim to simplify system design, http://www.electronics-eetimes.com/en/psoc-power-monitoring-and-fan-control-solutions-aim-to-simplify-system-design.html?cmp_id=7&news_id=222913088, fecha de consulta Agosto 2014.

[3] Terasic, Terasic, http://www.terasic.com.tw/en/, fecha de consulta julio 2014.

[4] INDTI INTI, Simposio Argentino de Sistemas Embebidos-SASE2012, http://www.inti.gob.ar/noticias/slide_simposio.htm, fecha de consulta agosto 2014.

[5] eStuffz, DE0-NANO easierpinout, https://sites.google.com/site/fpgaandco/de0-nano-pinout, fecha de consulta junio 2014.

[6] N. Instrument, FPGAs a Fondo, http://www.ni.com/white-paper/6983/es/, fecha de consulta septiembre 2014.

[7] K. Parnel y N. Metha, ProgrammableLoficDesign Quick StartHandbook, http://te.ugm.ac.id/~enas/tpe/temu6/BeginnnersBook-screen.pdf, fecha de consulta semptiembre 2014.

[8] Altera, DE0-NANO User Manual v.1.9,https://www.altera.com/en_US/pdfs/literature/ug/DE0_Nano_User_Manual_v1.9.pdf, fecha de consulta octubre 2014.

[9] Altera, NIOS II ClassicProcessor Reference Guide, https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/nios2/n2cpu_nii5v1.pdf, fecha de consulta octubre 2014.

[10] Altera, NIOS Processor: TheWorld'sMostVersatileEmbeddedProcessor, https://www.altera.com/products/processors/overview.html, fecha de consulta julio 2014.

[11] Cayssials, Ricardo, Libro de Sistemas Embebidos en FPGA, MARCOMBO S.A, 2014.

[12] Altera, NIOS II Software Build Tools, https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/nios2/n2sw_nii52015.pdf, fecha de consulta noviembre 2014.

[13] Altera, Nios II EmbebeddedDesign Suite, https://www.altera.com/products/design-software/embedded-software-developers/nios-ii-eds.html, fecha de consulta diciembre 2014.

[14] Pagel, Sigfredo, Síntesis Digital Directa de Frecuencias, DDFS, Revista Española de Electrónica A, 2007.

[15] M.Sc. Gonzalo, Macías, UNIDAD EN ARQUITECTURA FPGA MODULAR Y ACOPLE A SISTEMAS CIENTÍFICOS Y TECNOLÓFICOS QUE REQUERAN ANÁLISIS DE FASE Y AMPLITUD, Centro Universitario Querétaro, México 2011, fecha de consulta septiembre 2014.

[16] OpenCores.org, WaveformGen/NCO, http://opencores.org/project,waveform_gen, fecha de consulta julio 2014.

[17] OpenCores.org, WaveformGen/NCO Datasheet, http://www.zipcores.com/opencores/waveform_gen.pdf, fecha de consulta junio 2014.

[18] A.Devices, AD767 DIP, https://octopart.com/ad767ad-analog+devices-435917, fecha de consulta julio 2014.

[19] A. Devices, Microprocessor-Compatible 12 Bit D/A AD767, http://www.analog.com/media/en/technical-documentation/data-sheets/AD767.pdf, fecha de consulta agosto 2014.

[20] Yiying, Micriμm: μC/OS-II Real-Time Operating System Kernel Overview, http://www.element14.com/community/docs/DOC-46384/l/micri%C2%B5m-%C2%B5cos-ii-real-time-operating-system-kernel-overview, fecha de consultaoctubre 2014.

[21] Anónimo, Equipo Generador de Señales,http://www.ele.uva.es/~lourdes/docencia/Master_IE/2Equipos%20generadores%20de%20se%C3%B1al.pdf, fecha de consulta septiembre 2014.

[22] E-POS, TenconlogíaTouchScreen, http://www.e-postechnology.com/index.php/novedades-tecnologicas.html, fecha de consulta septiembre 2014.

[23] Arilla, C., Arriba, L., Tendencias y aplicaciones de los Sistemas Embebidos en España Estudio de Prospectiva, http://www.opti.org/publicaciones/pdf/texto131.pdf, fecha de consulta agosto 2014.

[24] Becerra, H., Diseño Avanzado de Hardware (ECE31289) 2015-I, https://sites.google.com/site/ece31289upb2015/practicas-de-clase/practica-7-rtos, fecha de consulta septiembre 2014.

[25] Enciclopedia Libre, OpenCores, http://es.wikipedia.org/wiki/OpenCores, fecha de consulta agosto 2014.

[26] Becerra, H, PRÁCTICA EMPRESARIAL EN UN BIOINGENIERÍA DE LA FUNDACIÓN CARDIOVASCULAR DE COLOMBIA, UNIVERSIDAD PONTIFICIA BOLIVARIANA, Bucaramanga 2011.

ANEXOS

ANEXO A: CÓDIGO DE FUENTE

main.c

```
#include <stdio.h>
#include <unistd.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

#include "includes.h"
#include "altera_avalon_pio_regs.h"
#include "system.h"
#include "io.h"
#include "alt_types.h"
#include "altera_avalon_spi_regs.h"
#include "terasic_includes.h"
#include "alt_touchscreen/alt_touchscreen.h"
#include "alt_touchscreen/touch_handlers.h"
#include "pantalla_eventos/eventos.h"

#define TOUCHSCREEN_SAMPLE_RATE 20 // 50 Sample-rate for touch
screen (Hz)
#define INITIAL_TIME 10            // Initial time (seconds)


/* Definition of Task Stacks */
#define   TASK_STACKSIZE       2048
OS_STK    task1_stk[TASK_STACKSIZE];
OS_STK    task2_stk[TASK_STACKSIZE];

/* Definition of Task Priorities */

#define TASK1_PRIORITY      1
#define TASK2_PRIORITY      2

VIP_FRAME_READER *display;

void actualizar_pantalla(void *datos){

      touch_control *touch = (touch_control *)datos;
      int pen_down=0, x, y;
      int prev_down = 0;
      display =  VIPFR_Init(ALT_VIP_VFR_BASE, (void *)FR_FRAME_0,
(void *)FR_FRAME_0, FRAME_WIDTH, FRAME_HEIGHT);
      VIPFR_Go(display, TRUE); // EDITADO
      VIPFR_ActiveDrawFrame(display);
      vid_clean_screen(display, WHITE_24);
```

```
      dibuja_gui(touch, display);


      while (1){
            alt_touchscreen_get_pen(&(touch->touchscreen),
&pen_down,&x,&y);

            if(pen_down==1 && prev_down == 0 && touch-
>evento==UI_PRESIONADO){
                  dibuja_gui(touch, display);
                  cambios_pantalla(touch, display);
                  prev_down = 1;

            }else if (pen_down==0 && touch->evento==UI_SOLTADO){
                  dibuja_gui(touch, display);
                  cambios_pantalla(touch, display);
                  mostrar_valores(touch, display);
                  touch->evento=UI_SIN_ACCION;
                  prev_down = 0;
            }

            OSTimeDlyHMSM(0, 0, 0, 17);
  }
}

void actualizar_tactil(void *datos){
        touch_control *touch = (touch_control *)datos;
        /*
         * Periodically check for registered events. The
         * alt_touchscreen_event_loop_update routine will
         * call the registered handlers, which execute as
         * part of this task.
         */
        while(1) {
          OSTimeDlyHMSM(0, 0, 0, (1000 / TOUCHSCREEN_SAMPLE_RATE));
          alt_touchscreen_event_loop_update(&(touch->touchscreen));
        }
}

int inicializar_dispositivos(touch_control *touch){
  int result = 0;
  IOWR_ALTERA_AVALON_SPI_CONTROL(SPI_TOUCH_CONTROLLER_BASE, 0);
  /*
   * Touch screen interface
   */
  result = alt_touchscreen_init(&touch->touchscreen,

      SPI_TOUCH_CONTROLLER_BASE,

      SPI_TOUCH_CONTROLLER_IRQ,
                                              TOUCH_IRQN_BASE,

      TOUCHSCREEN_SAMPLE_RATE,
```

```
      ALT_TOUCHSCREEN_SWAP_XY);

  /* Set calibration constants */
  alt_touchscreen_calibrate_upper_right (&touch->touchscreen,
                    3946,    3849,     // ADC readings
                     479,       0   ); // pixel coords

  alt_touchscreen_calibrate_lower_left  (&touch->touchscreen,
                    132,     148,      // ADC readings
                      0,     271   );  // pixel coords

  /*
   * Register event handlers. For the picture viewer, we care only
   * about pen-down events (for now). Additional handlers can be
   * added should this be enhanced to support a drag-to-zoom feature
   */
   result |= alt_touchscreen_register_callback_func(
     &touch->touchscreen,                 // the screen
     ALT_TOUCHSCREEN_CALLBACK_ON_PEN_DOWN,     // even to register
     ui_pen_down_handler,                      // routine to call
     touch);                              // context for routine

   result |= alt_touchscreen_register_callback_func(
      &touch->touchscreen,                 // the screen
      ALT_TOUCHSCREEN_CALLBACK_ON_PEN_UP,     // even to register
      ui_pen_up_handler,                       // routine to call
      touch);                              // context for routine

  if(result)
    return -1;

  /* Success */
  return 0;
}

int crear_tareas(touch_control *touch){
  INT8U result = 0;

  result |= OSTaskCreateExt((void *)actualizar_pantalla,
                            (void *)touch,//null
                            (void *)&task1_stk[TASK_STACKSIZE],
                            TASK1_PRIORITY,
                            TASK1_PRIORITY,
                            task1_stk,
                            TASK_STACKSIZE,
                            NULL,
                            0);

  result |= OSTaskCreateExt((void *)actualizar_tactil,
                                        (void *)touch,
                                      (void
*)&task2_stk[TASK_STACKSIZE],
```

```
                                        TASK2_PRIORITY,
                                        TASK2_PRIORITY,
                                        task2_stk,
                                        TASK_STACKSIZE,
                                        NULL,
                                        0);

  if(result == OS_NO_ERR) {
    printf("Success.\n");
  }else {
    printf("Failed.\n");
    return -1;
  }

  return 0;
}

/* The main function creates two task and starts multi-tasking */
int main(void){

      touch_control touch;
      touch.datos.generar = 0;
      touch.ui_command = 0;
      touch.timer = INITIAL_TIME * 10;
      touch.parametros.multiplicador=UI_X_1;
      touch.parametros.tipo_frec=UI_HZ;
      touch.parametros.tipo_onda=UI_SENO;
      touch.datos.amplitud=100;
      touch.datos.frecuencia=100;
      touch.evento=UI_SIN_ACCION;

      if(inicializar_dispositivos(&touch)) {
            printf("\nError initializing IO devices. Halting.\n");
            return -1;
      }

      if(crear_tareas(&touch)) {
            printf("\nError creating MicroC/OS-II tasks. Halting\n");
            return -1;
      }

   OSStart();
  return 0;

}
```

eventos.c

```
/*
 * eventos.c
 *
 * Created on: 29/04/2015
 *      Author: Roberto Noboa - Sergio Chamba
       * Descripción: Procedimientos que realizan las funcionalidades
      requeridas del generador y la pantalla
/
#include <stdio.h>
#include "eventos.h"

void cambios_pantalla(touch_control *touch, VIP_FRAME_READER
*display){

      if(touch->ui_command==UI_MENOS_AMP)
            DispIMAGE((void*)display, 40, 47,&BTN_ATRAS_2);
      else if (touch->ui_command==UI_MAS_AMP)
            DispIMAGE((void*)display, 170, 47,&BTN_ADELANTE_2);

      else if (touch->ui_command==UI_MENOS_FREC)
            DispIMAGE((void*)display, 205, 47,&BTN_ATRAS_2);
      else if (touch->ui_command==UI_MAS_FREC)
            DispIMAGE((void*)display, 332, 47,&BTN_ADELANTE_2);

      else if (touch->ui_command==0){
            DispIMAGE((void*)display, 170, 47,&BTN_ADELANTE);
            DispIMAGE((void*)display, 40, 47,&BTN_ATRAS);
            DispIMAGE((void*)display, 332, 47,&BTN_ADELANTE);
            DispIMAGE((void*)display, 205, 47,&BTN_ATRAS);
      }

}

void mostrar_valores(touch_control *touch, VIP_FRAME_READER
*display){
      mostrar_amplitud(touch, display);
      mostrar_frecuencia(touch, display);
}

void mostrar_amplitud(touch_control *touch, VIP_FRAME_READER
*display){
      char valor_pantalla[8];
      snprintf(valor_pantalla,8,"%d", touch->datos.amplitud);
      vid_draw_box(100,53,130,63,WHITE_24,DO_FILL,display);
      vid_print_string(100, 53, SLATEBLUE_24, cour10_font, display,
valor_pantalla);
}

void mostrar_frecuencia(touch_control *touch, VIP_FRAME_READER
*display){
```

```
      char valor_pantalla[8];
      float frec_pantalla = (touch->datos.frecuencia);
      if(touch->parametros.tipo_frec==UI_HZ)
            snprintf(valor_pantalla,8,"%.1f",frec_pantalla);
      else if(touch->parametros.tipo_frec==UI_KHZ)
            snprintf(valor_pantalla,8,"%.4f",(frec_pantalla/1000));
      else if(touch->parametros.tipo_frec==UI_MHZ)

      snprintf(valor_pantalla,8,"%.4f",(frec_pantalla/1000000));
      vid_draw_box(260,53,315,63,WHITE_24,DO_FILL,display);
      vid_print_string(260, 53, SLATEBLUE_24, cour10_font, display,
valor_pantalla);
}


void dibuja_gui(touch_control *touch, VIP_FRAME_READER *display){

      VIPFR_ActiveDrawFrame(display);

      DispIMAGE((void*)display, 0, -8,&ESPOL);

    /* AMPLITUD */
    DispIMAGE((void*)display, 40, 47,&BTN_ATRAS);
    DispIMAGE((void*)display, 62, 40,&AMPLITUD);
    DispIMAGE((void*)display, 170, 47,&BTN_ADELANTE);

    /* FRECUENCIA */
    DispIMAGE((void*)display, 205, 47,&BTN_ATRAS);
    DispIMAGE((void*)display, 227, 40,&FRECUENCIA);
    DispIMAGE((void*)display, 332, 47,&BTN_ADELANTE);

    /* CONTROLES */
    if (touch->parametros.tipo_frec==UI_HZ)
      DispIMAGE((void*)display, 62, 80,&BTN_Hz_2);
    else
      DispIMAGE((void*)display, 62, 80,&BTN_Hz);

    if (touch->parametros.tipo_frec==UI_KHZ)
            DispIMAGE((void*)display, 185, 80,&BTN_KHZ_2);
    else
            DispIMAGE((void*)display, 185, 80,&BTN_KHZ);

    DispIMAGE((void*)display, 307, 80,&BTN_MHZ_3);

    if (touch->parametros.multiplicador==UI_X_1)
      DispIMAGE((void*)display, 62, 115,&BTN_X1_2);
    else
      DispIMAGE((void*)display, 62, 115,&BTN_X1);

    if (touch->parametros.multiplicador==UI_X_10)
            DispIMAGE((void*)display, 185, 115,&BTN_X10_2);
      else
            DispIMAGE((void*)display, 185, 115,&BTN_X10);
```

```
    if (touch->parametros.multiplicador==UI_X_100)
      DispIMAGE((void*)display, 307, 115,&BTN_X100_2);
      else
            DispIMAGE((void*)display, 307, 115,&BTN_X100);

    DispIMAGE((void*)display, 173, 145,&TIPO_ONDA);

    if (touch->parametros.tipo_onda==UI_SENO)
            DispIMAGE((void*)display, 51, 155,&BTN_SENO_2);
      else
            DispIMAGE((void*)display, 51, 155,&BTN_SENO);

    if (touch->parametros.tipo_onda==UI_CUADRADO)
            DispIMAGE((void*)display, 132, 155,&BTN_CUADRADA_2);
      else
            DispIMAGE((void*)display, 132, 155,&BTN_CUADRADA);

    if (touch->parametros.tipo_onda==UI_TRIANGULAR)
            DispIMAGE((void*)display, 213, 155,&BTN_TRIANGULAR_2);
      else
            DispIMAGE((void*)display, 213, 155,&BTN_TRIANGULAR);

    if (touch->parametros.tipo_onda==UI_SIERRA)
            DispIMAGE((void*)display, 299, 155,&BTN_SIERRA_2);
      else
            DispIMAGE((void*)display, 299, 155,&BTN_SIERRA);

    if (touch->datos.generar==0){
      DispIMAGE((void*)display, 379, 160,&BTN_GENERAR);
      DispIMAGE((void*)display, 379, 195,&BTN_STOP_2);
    }else{
      DispIMAGE((void*)display, 379, 160,&BTN_GENERAR_2);
      DispIMAGE((void*)display, 379, 195,&BTN_STOP);
    }

    mostrar_valores(touch,display);
    presentar_info(touch,display);

}

void presentar_info(touch_control *touch, VIP_FRAME_READER
*display){
      char mensajes[100];
      int bandera = 0;
      int bandera2 = 0;

      /* ESCENARIOS DONDE SE INCIA LA DISTORCIÓN DE LA ONDA Y OK */
      if (touch->parametros.tipo_onda==UI_SENO && touch-
>datos.frecuencia >= 550000){
            bandera = 1;

            if (touch->informacion==INFO_MAX_VOLTAJE)
```

```
                        snprintf(mensajes,100,"V.:%.2f [V] | Alerta:
Distorcion de onda."

                  "| Info.: Vmax.: 10 [V]", (touch-
>datos.amplitud/10));
            if (touch->informacion==INFO_MIN_VOLTAJE)
                        snprintf(mensajes,100,"V.:%.2f [V] |
Alerta: Distorcion de onda."

                        "| Info.: Vmin.: 0.1 [V]", (touch-
>datos.amplitud/10));
            if (touch->informacion==SIN_INFO)
                  snprintf(mensajes,100,"V.:%.2f [V] | Alerta:
Distorcion de onda."
                                                                  "|
Info.: OK", (touch->datos.amplitud/10));
      }

      if ((touch->parametros.tipo_onda==UI_CUADRADO || touch-
>parametros.tipo_onda==UI_TRIANGULAR)
             && touch->datos.frecuencia >= 300000){
            bandera = 1;

            if (touch->informacion==INFO_MAX_VOLTAJE)
                        snprintf(mensajes,100,"V.:%.2f [V] | Alerta:
Distorcion de onda."

                  "| Vmax.: 10 [V]", (touch->datos.amplitud/10));
            if (touch->informacion==INFO_MIN_VOLTAJE)
                        snprintf(mensajes,100,"V.:%.2f [V] | Alerta:
Distorcion de onda."

                        "| Vmin.: 0.1 [V]", (touch-
>datos.amplitud/10));
            if (touch->informacion==SIN_INFO)
                  snprintf(mensajes,100,"V.:%.2f [V] | Alerta:
Distorcion de onda."

            "| Info.: OK", (touch->datos.amplitud/10));
      }

      if (touch->parametros.tipo_onda==UI_SIERRA && touch-
>datos.frecuencia >= 180000){
            bandera = 1;

            if (touch->informacion==INFO_MAX_VOLTAJE)
                        snprintf(mensajes,100,"V.:%.2f [V] | Alerta:
Distorcion de onda."

                  "| Vmax.: 10 [V]", (touch->datos.amplitud/10));
            if (touch->informacion==INFO_MIN_VOLTAJE)
                        snprintf(mensajes,100,"V.:%.2f [V] | Alerta:
Distorcion de onda."
```

```
                        "| Vmin.: 0.1 [V]", (touch-
>datos.amplitud/10));
            if(touch->informacion==SIN_INFO)
                  snprintf(mensajes,100,"V.:%.2f [V] | Alerta:
Distorcion de onda."

            "| Info.: OK", (touch->datos.amplitud/10));
      }


      /* ESCENARIOS DONDE SE INFORMA VALIDACIÓN DE LAS FRECUENCIAS */
      if (touch->informacion==INFO_MAX_FRECUENCIA)
            snprintf(mensajes,100,"V.:%.2f [V] | Alerta: Onda
distorcionada."

                        "| Fmax.: 2 [Mhz].", (touch-
>datos.amplitud/10));
      if (touch->informacion==INFO_MIN_FRECUENCIA)
            snprintf(mensajes,100,"V.:%.2f [V] | Alerta: Ninguna."
                                                            "|
Fmin.: 1 [Hz].", (touch->datos.amplitud/10));
      if(touch->informacion==SIN_INFO && bandera == 0){
            bandera2 = 1;
            snprintf(mensajes,100,"V.:%.2f [V] | Alerta: Ninguna."
                                          "| Info.: OK.", (touch-
>datos.amplitud/10));
      }

      /* ESCENARIOS DONDE SE INFORMA VALIDACIÓN DE LAS AMPLITUD */
      if (touch->informacion==INFO_MAX_VOLTAJE && bandera == 0)
            snprintf(mensajes,100,"V.:%.2f [V] | Alerta: Ninguna."

                        "| Vmax.: 10 [V].", (touch-
>datos.amplitud/10));
      if (touch->informacion==INFO_MIN_VOLTAJE && bandera == 0)
            snprintf(mensajes,100,"V.:%.2f [V] | Alerta: Ninguna."

      "| Vmin.: 0.1 [V].", (touch->datos.amplitud/10));

      vid_draw_box(0,260,480,275,WHITE_24,DO_FILL,display);

      if (bandera2 == 1)
            vid_print_string(5, 260, SLATEBLUE_24, cour10_font,
display, mensajes);
      else
            vid_print_string(5, 260, RED_24, cour10_font, display,
mensajes);

      bandera = 0;
      bandera2 = 0;

}
```

eventos.h

```
/*
 * eventos.h
 *
 *Created on: 29/04/2015
 *Author: Roberto Noboa - Sergio Chamba
       *Descripciones: Librerías y esctructuras requeridas para el
      desarrollo del proyecto
*/
#ifndef EVENTOS_H_
#define EVENTOS_H_

#include <stdio.h>
#include <unistd.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

#include "includes.h"
#include "altera_avalon_pio_regs.h"
#include "system.h"
#include "../vip_fr.h"
#include "../graphic_lib/simple_graphics.h"
#include "../graphic_lib/gimp_bmp.h"
#include "../graphic_lib/draw_gimps.h"
#include "../terasic_includes.h"
#include "../alt_touchscreen/touch_handlers.h"

#define FRAME_WIDTH  480
#define FRAME_HEIGHT 272

#define FR_FRAME_0  (SDRAM_BASE)
#define FR_FRAME_1  (SDRAM_BASE + FRAME_WIDTH*FRAME_HEIGHT*4)

extern struct gimp_image_struct ESPOL;
extern struct gimp_image_struct BTN_ATRAS;
extern struct gimp_image_struct BTN_ATRAS_2;
extern struct gimp_image_struct BTN_ADELANTE;
extern struct gimp_image_struct BTN_ADELANTE_2;
extern struct gimp_image_struct BTN_SENO;
extern struct gimp_image_struct BTN_SENO_2;
extern struct gimp_image_struct BTN_CUADRADA;
extern struct gimp_image_struct BTN_CUADRADA_2;
extern struct gimp_image_struct BTN_TRIANGULAR;
extern struct gimp_image_struct BTN_TRIANGULAR_2;
extern struct gimp_image_struct BTN_SIERRA;
extern struct gimp_image_struct BTN_SIERRA_2;
extern struct gimp_image_struct BTN_GENERAR;
extern struct gimp_image_struct BTN_GENERAR_2;
extern struct gimp_image_struct BTN_STOP;
```

```
extern struct gimp_image_struct BTN_STOP_2;

extern struct gimp_image_struct BTN_VER;
extern struct gimp_image_struct BTN_VER_2;
extern struct gimp_image_struct BTN_VER_3;


extern struct gimp_image_struct FRECUENCIA;
extern struct gimp_image_struct TIPO_ONDA;
extern struct gimp_image_struct AMPLITUD;



extern struct gimp_image_struct BTN_Hz;
extern struct gimp_image_struct BTN_Hz_2;

extern struct gimp_image_struct BTN_KHZ;
extern struct gimp_image_struct BTN_KHZ_2;
extern struct gimp_image_struct BTN_MHZ;
extern struct gimp_image_struct BTN_MHZ_2;
extern struct gimp_image_struct BTN_MHZ_3;

extern struct gimp_image_struct BTN_X1;
extern struct gimp_image_struct BTN_X1_2;
extern struct gimp_image_struct BTN_X10;
extern struct gimp_image_struct BTN_X10_2;
extern struct gimp_image_struct BTN_X100;
extern struct gimp_image_struct BTN_X100_2;
extern struct gimp_image_struct BTN_SIGUIENTE_2;

void cambios_pantalla(touch_control *touch, VIP_FRAME_READER
*display);
void dibuja_gui(touch_control *touch, VIP_FRAME_READER *display);
void mostrar_valores(touch_control *touch, VIP_FRAME_READER
*display);
void mostrar_amplitud(touch_control *touch, VIP_FRAME_READER
*display);
void mostrar_frecuencia(touch_control *touch, VIP_FRAME_READER
*display);

#endif /* EVENTOS_H_ */
```

touch_handlers.c

```
#include <stdio.h>
#include "system.h"
#include "touch_handlers.h"
#include <math.h>

volatile int *Enable_GEN = (int *)ENABLE_GEN_BASE;
volatile int *ResetN = (int *)RESET_GEN_BASE;
volatile int *Phase = (int *)FRECUENCIA_IN_BASE;
volatile int *Amplitud = (int *)AMPLITUD_BASE;
volatile int *Select = (int *)SELECTOR_GEN_BASE;

void ui_pen_down_handler(int pen_down, int x, int y, void* context){

	touch_control *touch = (touch_control *)(context);

	if(x>= MENOS_AMP_X_MIN && x <= MENOS_AMP_X_MAX &&
	 y>=MENOS_AMP_Y_MIN  && y <= MENOS_AMP_Y_MAX) {
		touch->ui_command = UI_MENOS_AMP;
		touch->evento = UI_PRESIONADO;
	}
	else if(x>= MAS_AMP_X_MIN && x <= MAS_AMP_X_MAX &&
			y>= MAS_AMP_Y_MIN && y <= MAS_AMP_Y_MAX) {
		touch->ui_command = UI_MAS_AMP;
		touch->evento = UI_PRESIONADO;
	}
	else if(x>= MENOS_FREC_X_MIN && x <= MENOS_FREC_X_MAX &&
			y>= MENOS_FREC_Y_MIN && y <= MENOS_FREC_Y_MAX) {
		touch->ui_command = UI_MENOS_FREC;
		touch->evento = UI_PRESIONADO;
	}
	else if(x>= MAS_FREC_X_MIN && x <= MAS_FREC_X_MAX &&
			y>= MAS_FREC_Y_MIN && y <= MAS_FREC_Y_MAX) {
		touch->ui_command = UI_MAS_FREC;
		touch->evento = UI_PRESIONADO;
	}
	else if(x>= GENERAR_X_MIN && x <= GENERAR_X_MAX &&
			y>= GENERAR_Y_MIN && y <= GENERAR_Y_MAX) {
		touch->ui_command = UI_GENERAR;
		touch->evento = UI_PRESIONADO;
	}
	else if(x>= STOP_X_MIN && x <= STOP_X_MAX &&
			y>= STOP_Y_MIN && y <= STOP_X_MAX) {
		touch->ui_command = UI_STOP;
		touch->evento = UI_PRESIONADO;
	}
	else if(x>= X1_X_MIN && x <= X1_X_MAX &&
			y>= X1_Y_MIN && y <= X1_Y_MAX) {
		touch->parametros.multiplicador = UI_X_1;
		touch->evento = UI_PRESIONADO;
	}
```

```
        else if(x>= X10_X_MIN && x <= X10_X_MAX &&
                y>= X10_Y_MIN && y <= X10_Y_MAX) {
              touch->parametros.multiplicador = UI_X_10;
              touch->evento = UI_PRESIONADO;
        }
        else if(x>= X100_X_MIN && x <= X100_X_MAX &&
                y>= X100_Y_MIN && y <= X100_Y_MAX) {
              touch->parametros.multiplicador = UI_X_100;
              touch->evento = UI_PRESIONADO;
        }
        else if(x>= SENO_X_MIN && x <= SENO_X_MAX &&
                y>= SENO_Y_MIN && y <= SENO_Y_MAX) {
              touch->parametros.tipo_onda = UI_SENO;
              touch->evento = UI_PRESIONADO;
              *Select = 0x00;
        }
        else if(x>= CUADRADO_X_MIN && x <= CUADRADO_X_MAX &&
                y>= CUADRADO_Y_MIN && y <= CUADRADO_Y_MAX) {
              touch->parametros.tipo_onda = UI_CUADRADO;
              *Select = 0x03;
              touch->evento = UI_PRESIONADO;
        }
        else if(x>= TRIANGULAR_X_MIN && x <= TRIANGULAR_X_MAX &&
                y>= TRIANGULAR_Y_MIN && y <= TRIANGULAR_Y_MAX) {
              touch->parametros.tipo_onda = UI_TRIANGULAR;
              *Select = 0x01;
              touch->evento = UI_PRESIONADO;
        }
        else if(x>= SIERRA_X_MIN && x <= SIERRA_X_MAX &&
                y>= SIERRA_Y_MIN && y <= SIERRA_Y_MAX) {
              touch->parametros.tipo_onda = UI_SIERRA;
              *Select = 0x02;
              touch->evento = UI_PRESIONADO;
        }
        else if(x>= HZ_X_MIN && x <= HZ_X_MAX &&
                y>= HZ_Y_MIN && y <= HZ_Y_MAX) {
              touch->parametros.tipo_frec = UI_HZ;
              touch->ui_command = UI_HZ;
              touch->evento = UI_PRESIONADO;
        }
        else if(x>= KHZ_X_MIN && x <= KHZ_X_MAX &&
                y>= KHZ_Y_MIN && y <= KHZ_Y_MAX) {
              touch->parametros.tipo_frec = UI_KHZ;
              touch->ui_command = UI_KHZ;
              touch->evento = UI_PRESIONADO;
        }
        else if(x>= MHZ_X_MIN && x <= MHZ_X_MAX &&
                y>= MHZ_Y_MIN && y <= MHZ_Y_MAX) {
              touch->parametros.tipo_frec = UI_MHZ;
              touch->ui_command = UI_MHZ;
              touch->evento = UI_PRESIONADO;
        }
}
```

```
void ui_pen_up_handler (int pen_up, int x, int y, void* context){
      float Fmin, Fmax, Fs=20000000;
      int Amin, Amax;
      Fmin= Fs/pow(2,32);
      Fmax=Fs/10;
      Amin=1;
      Amax=100;

    touch_control *touch = (touch_control *)(context);
    touch->informacion = SIN_INFO;

    if(touch->ui_command==UI_MENOS_AMP){
            touch->ui_command=0;
            if(touch->datos.amplitud > Amin){
                  switch (touch->parametros.multiplicador){
                        case UI_X_1:
                              if(((touch->datos.amplitud)-1) >= Amin)
                                    (touch->datos.amplitud)-=1;
                              else
                                    touch->informacion =
INFO_MIN_VOLTAJE;
                              break;
                        case UI_X_10:
                              if(((touch->datos.amplitud)-10) >=
Amin)
                                    (touch->datos.amplitud)-=10;
                              else
                                    touch->informacion =
INFO_MIN_VOLTAJE;
                              break;
                        case UI_X_100:
                              if(((touch->datos.amplitud)-100) >=
Amin)
                                    (touch->datos.amplitud)-=100;
                              else
                                    touch->informacion =
INFO_MIN_VOLTAJE;
                              break;
                  }
            }
            else
                  touch->informacion = INFO_MIN_VOLTAJE;
      }
      else if (touch->ui_command==UI_MAS_AMP){
            touch->ui_command=0;

            if(touch->datos.amplitud < Amax){
                  switch (touch->parametros.multiplicador){
                        case UI_X_1:
                              if(((touch->datos.amplitud)+1) <= Amax)
                                    (touch->datos.amplitud)+=1;
                              else
```

```
                                    touch->informacion =
INFO_MAX_VOLTAJE;
                              break;
                        case UI_X_10:
                              if(((touch->datos.amplitud)+10) <=
Amax)
                                    (touch->datos.amplitud)+=10;
                              else
                                    touch->informacion =
INFO_MAX_VOLTAJE;
                              break;
                        case UI_X_100:
                              if(((touch->datos.amplitud)+100) <=
Amax)
                                    (touch->datos.amplitud)+=100;
                              else
                                    touch->informacion =
INFO_MAX_VOLTAJE;
                              break;
                  }
            }else
                  touch->informacion = INFO_MAX_VOLTAJE;
      }
      else if (touch->ui_command==UI_MENOS_FREC){
            touch->ui_command=0;

            if(touch->datos.frecuencia <= Fmin)
                  touch->informacion = INFO_MIN_FRECUENCIA;

            else{

                  switch (touch->parametros.tipo_frec){
                        case UI_HZ:
                              switch (touch-
>parametros.multiplicador){
                                    case UI_X_1:
                                          if(((touch-
>datos.frecuencia)-1) >= Fmin)
                                                (touch-
>datos.frecuencia)-=1;
                                          else
                                                touch->informacion =
INFO_MIN_FRECUENCIA;
                                          break;
                                    case UI_X_10:
                                          if(((touch-
>datos.frecuencia)-10) >= Fmin)
                                                (touch-
>datos.frecuencia)-=10;
                                          else
                                                touch->informacion =
INFO_MIN_FRECUENCIA;
                                          break;
```

```
                                        case UI_X_100:
                                              if(((touch-
>datos.frecuencia)-100) >= Fmin)
                                                    (touch-
>datos.frecuencia)-=100;
                                              else
                                                    touch->informacion =
INFO_MIN_FRECUENCIA;
                                              break;
                                  }
                                  break;

                            case UI_KHZ:
                                  switch (touch-
>parametros.multiplicador){
                                        case UI_X_1:
                                              if(((touch-
>datos.frecuencia)-1000) >= Fmin)
                                                    (touch-
>datos.frecuencia)-=1000;
                                              else
                                                    touch->informacion =
INFO_MIN_FRECUENCIA;
                                              break;
                                        case UI_X_10:
                                              if(((touch-
>datos.frecuencia)-10000) >= Fmin)
                                                    (touch-
>datos.frecuencia)-=10000;
                                              else
                                                    touch->informacion =
INFO_MIN_FRECUENCIA;
                                              break;
                                        case UI_X_100:
                                              if(((touch-
>datos.frecuencia)-100000) >= Fmin)
                                                    (touch-
>datos.frecuencia)-=100000;
                                              else
                                                    touch->informacion =
INFO_MIN_FRECUENCIA;
                                              break;
                                  }
                                  break;

                            case UI_MHZ:
                                  switch (touch-
>parametros.multiplicador){
                                        case UI_X_1:
                                              if(((touch-
>datos.frecuencia)-1000000) >= Fmin)
                                                    (touch-
>datos.frecuencia)-=1000000;
```

```
                                                      else
                                                            touch->informacion =
INFO_MIN_FRECUENCIA;
                                                      break;
                                          }
                                    break;
                              }
                        }
                  }

                  else if (touch->ui_command==UI_MAS_FREC){
                        touch->ui_command=0;

                        if(touch->datos.frecuencia >= Fmax)
                              touch->informacion = INFO_MAX_FRECUENCIA;
                        else{
                              switch (touch->parametros.tipo_frec){
                                    case UI_HZ:
                                          switch (touch-
>parametros.multiplicador){
                                                case UI_X_1:
                                                      if(((touch-
>datos.frecuencia)+1) <= Fmax)
                                                            (touch-
>datos.frecuencia)+=1;
                                                      else
                                                            touch->informacion =
INFO_MAX_FRECUENCIA;
                                                      break;
                                                case UI_X_10:
                                                      if(((touch-
>datos.frecuencia)+10) <= Fmax)
                                                            (touch-
>datos.frecuencia)+=10;
                                                      else
                                                            touch->informacion =
INFO_MAX_FRECUENCIA;
                                                      break;
                                                case UI_X_100:
                                                      if(((touch-
>datos.frecuencia)+100) <= Fmax)
                                                            (touch-
>datos.frecuencia)+=100;
                                                      else
                                                            touch->informacion =
INFO_MAX_FRECUENCIA;
                                                      break;
                                          }
                                          break;
                                    case UI_KHZ:
                                          switch (touch-
>parametros.multiplicador){
                                                case UI_X_1:
```

```
                                                if(((touch-
>datos.frecuencia)+1000) <= Fmax)
                                                      (touch-
>datos.frecuencia)+=1000;
                                                else
                                                      touch->informacion =
INFO_MAX_FRECUENCIA;
                                                break;
                                          case UI_X_10:
                                                if(((touch-
>datos.frecuencia)+10000) <= Fmax)
                                                      (touch-
>datos.frecuencia)+=10000;
                                                else
                                                      touch->informacion =
INFO_MAX_FRECUENCIA;
                                                break;
                                          case UI_X_100:
                                                if(((touch-
>datos.frecuencia)+100000) <= Fmax)
                                                      (touch-
>datos.frecuencia)+=100000;
                                                else
                                                      touch->informacion =
INFO_MAX_FRECUENCIA;
                                                break;
                                    }
                                    break;
                              case UI_MHZ:
                                    switch (touch-
>parametros.multiplicador){
                                          case UI_X_1:
                                                if(((touch-
>datos.frecuencia)+1000000) <= Fmax)
                                                      (touch-
>datos.frecuencia)+=1000000;
                                                else
                                                      touch->informacion =
INFO_MAX_FRECUENCIA;
                                                break;
                                    }
                              break;
                        }
                  }
            }

      else if (touch->ui_command==UI_GENERAR){
                  touch->ui_command=0;
                  touch->datos.generar=1;
                  touch->datos.incrementado =(((touch-
>datos.frecuencia)*pow(2,32))/Fs) + 0.5;
                  generarHardware(touch);
            }
```

```
      else if (touch->ui_command==UI_STOP){
            touch->ui_command=0;
            touch->datos.generar=0;
            detenerHardware();
      }

   touch->evento = UI_SOLTADO;

}

void generarHardware(touch_control *touch){
      *ResetN = 1;
      *Enable_GEN = 1;
      *Phase = (int)(touch->datos.incrementado);
      *Amplitud = (int)(touch->datos.amplitud);
}

void detenerHardware(){
      *ResetN=0;
      *Enable_GEN=0;
}
```