ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"MÁQUINA CALCULADORA BASADA EN EL ALGORITMO CORDIC CON NIOS II"

TESINA DE SEMINARIO

Previa la obtención del Título de:

INGENIERO EN TELEMÀTICA

Presentado por:

Basurto Chicango Juliana Abigail

Crespín Cedeño Jorge Xavier

GUAYAQUIL - ECUADOR

AÑO 2013

AGRADECIMIENTO

Quiero agradecer primero a Dios porque Él me ha provisto de todo lo necesario para llegar hasta esta meta, a mis padres por su apoyo invaluable, a mi compañero y excelente amigo Jorge Crespín por su arduo trabajo en este proyecto, al profesor y amigo MSC. Ronald Ponguillo por su orientación y dedicación brindada para el desarrollo de este proyecto, por último quiero expresar mi agradecimiento a las personas que de alguna manera aportaron con sus conocimientos.

Juliana Abigail Basurto Chicango

Debo agradecer de manera muy especial al ser que me supo enseñar desde pequeño que la manera más difícil y a la vez honesta que nos permitiría superar la pobreza es estudiando, y hoy luego de tantos años de haber recibido esa lección puedo verla reflejada en este documento.

Hoy que aún te tengo conmigo quiero darte gracias mamá.

Jorge Xavier Crespín Cedeño

DEDICATORIA

A Dios,

Mis padres,

A quienes me brindaron su apoyo.

Juliana Abigail Basurto Chicango

A Dios,

Familia,

Y amigos.

Jorge Xavier Crespín Cedeño

TRIBUNAL DE SUSTENTACIÓN

MSc. Ronald Ponguillo

PROFESOR DE SEMINARIO DE GRADUACIÓN

MSc. Sara Ríos O.

PROFESOR DELEGADO POR EL DECANO DE LA FACULTAD

DECLARACIÓN EXPRESA

“La responsabilidad del contenido de esta tesina, 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)

Juliana Abigail Basurto Chicango

Jorge Xavier Crespín Cedeño

RESUMEN

El presente documento contiene toda la información sobre el desarrollo e implementación de una **MÁQUINA CALCULADORA BASADA EN EL ALGORITMO CORDIC CON NIOS II.**

La parte más importante en el desarrollo de esta máquina calculadora se centra en la aplicación de un **IP CORE** que permite el cálculo de las funciones Seno y Coseno de un valor de ángulo que es ingresado por el usuario. El **IP CORE** usa el algoritmo **CORDIC** para el cálculo de dichas funciones y por medio de la implementación de un bus Avalon se interconecta a un **PROCESADOR NIOS II INTEGRADO** y a otros bloques embebidos en un dispositivo **ALTERA CYCLONE II FPGA**.

Este escrito está estructurado en 4 capítulos que se detalla brevemente a continuación:

En el **primer capítulo**, se plantean la realización de los objetivos generales y específicos del proyecto, además se incluyen los alcances y limitaciones de su implementación, así como la justificación de su desarrollo.

En el **segundo capítulo**, se explican los fundamentos teóricos de los Sistemas en Chips Configurables y del algoritmo CORDIC, sus ventajas y usos en la tecnología actual.

En el **tercer capítulo**, se detalla el desarrollo e implementación de este proyecto en cada una de sus etapas.

En el **cuarto capítulo** se redacta el desarrollo y los resultados obtenidos de varias pruebas que se realizaron con el objetivo de comprobar la eficiencia del algoritmo CORDIC en comparación con otros métodos que realicen el cálculo de funciones seno y coseno.

Por último se muestra las conclusiones y recomendaciones de este proyecto.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ABREVIATURAS

ADC	Analog-to-Digital Converter
ASIC	Application Specific Integrated Circuit
CMOS	Complementary metal–oxide–semiconductor
CORDIC	Coordinate Rotation Digital Computer
CPLD	Complex Programmable Logic Device
DE2	Development and Education Board 2
DSP	Digital Signal Processing
E/S	Entrada y Salida
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input/Output
HDL	Hardware Description Language
JTAG	Joint Test Action Group
LCD	Liquid Crystal Display

LED	Light-Emitting Diode
PCB	Printed Circuit Board
RS-232	Recommended Standard 232
RISC	Reduced Instruction Set Computing
SOPC	System on a Programmable Chip
TTL	Transistor-Transistor Logic
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus

INTRODUCCIÓN

Uno de los tantos problemas presentes en la ingeniería es el cálculo de funciones trascendentales (funciones trigonométricas, logarítmicas y exponenciales) debido a su complejidad y a los recursos que se requiere para la implementación, en la actualidad existen grandes sistemas que usan aplicaciones que requieren de este tipo de cálculo, como ejemplo podemos mencionar a los detectores de fase, donde es necesario el cálculo de la tangente inversa para sincronización de portadora en sistemas de Radio Software (SR) o los sistemas de aviación donde debido a diferentes situaciones climáticas es necesario calcular la rapidez y velocidad de vuelo realizando funciones senos y cosenos.

Esta “dificultad” queda atrás gracias al desarrollo de tecnologías de circuitos digitales que tiene como uno de sus principales objetivos, realizar operaciones con el menor hardware posible, o lo que es aún mejor, que sobre el mismo hardware se pueda implementar más operaciones diferentes. Con la aparición de los FPGAs (*del inglés Field Programmable Gate Array*), hoy en día es posible programar un chip con un SOPC (Sistema en Chip Programable) y añadirle nuevas características si el flujo de su diseño cambia o reprogramar el chip con

un nuevo conjunto de instrucciones, con la ventaja de que no es necesario adquirir y/o añadir hardware adicional.

La enorme flexibilidad que brindan las FPGAs al flujo de diseño de circuitos digitales es la característica principal por la cual se han convertido en herramientas útiles de desarrollo e investigación para estudiantes de universidades y profesionales.

Cualquier circuito que tenga una aplicación específica puede ser implementado en un FPGA, siempre y cuando ésta disponga de los recursos necesarios.

Cabe notar que su uso en otras áreas es cada vez mayor, sobre todo en aquellas aplicaciones que requieren un alto grado de paralelismo, es decir, que el sistema implementado en la FPGAs pueda dar resultados en el menor tiempo posible.

El uso de un FPGA nos brinda una excelente vía para el desarrollo de aplicaciones que requieran del cálculo de funciones matemáticas (inclusive complejas) sobre ambientes de software donde no se posee unidades aritméticas complejas o multiplicadores, y sobre todo a bajo costo.

Este documento detalla el diseño e implementación de un SOPC en un dispositivo **ALTERA CYCLONE II FPGA** con **PROCESADOR NIOS II INTEGRADO,** capaz de realizar el cálculo de las funciones seno, coseno y tangente usando el algoritmo CORDIC, que es un conjunto de instrucciones basadas en desplazamientos, sumas y tablas de búsqueda, que no requiere de unidades aritméticas específicas para ser implementado, lo que lo hace muy efectivo en nuestro hardware.

CAPÍTULO 1

1 GENERALIDADES

En este capítulo se abarca el planteamiento del proyecto, se describe cuáles son sus objetivos; así como también se analiza cuáles son los alcances y limitaciones del mismo.

1.1 OBJETIVOS

1.1.1 OBJETIVOS GENERALES

Los objetivos generales a llevar a cabo en este trabajo son los siguientes:

- Agregar un bloque calculador de Senos y Cosenos basado en el algoritmo CORDIC[1] a un sistema basado en un Microprocesador NIOS II, a través del bus Avalon.

[1] El algoritmo CORDIC permite realizar el cálculo de las funciones trigonométricas, hiperbólicas, funciones logarítmicas, multiplicaciones reales y complejas, división, la raíz cuadrada, la solución de sistemas lineales, la estimación de valor propio, descomposición de valor singular, factorización QR de matrices [2],[6].

- Diseñar una máquina calculadora basada en el algoritmo CORDIC utilizando las herramientas de software Quartus II y Nios II e implantarla en un chip de la familia Cyclon II de Altera.

1.1.2 OBJETIVOS ESPECÍFICOS

- Estudiar los sistemas en Chips Configurables (SOPC – System on Programable Chip) y su implementación con las herramientas de software Quartus II y NIOS II en un chip de la familia Cyclon II de Altera.

- Añadir un IP Core que permita el cálculo de las funciones Senos y Cosenos usando el algoritmo CORDIC y diseñar una interfaz Avalon capaz de interconectar el IP Core antes mencionado con los demás bloques del sistema en chip.

- Comprobar la eficiencia del cálculo de funciones trascendentales usando el algoritmo CORDIC frente a otros métodos de cálculo en escenarios con condiciones semejantes de hardware.

- Implementar el proyecto, y en relación con los resultados obtener conclusiones y recomendaciones acerca de las pruebas realizadas.

- Proyectar el sistema en chip con bloque calculador de Senos y Cosenos a futuros proyectos e implementaciones.

1.2 ALCANCE Y LIMITACIONES DEL PROYECTO

Los alcances de este proyecto son:

- El ingreso de datos se realiza con un teclado hexadecimal 4x4, que está conectado a la tarjeta DE2 por medio de uno de sus puertos de expansión integrados (JP2).

- El teclado hexadecimal 4x4 posee teclas correspondientes a los números del 1 hasta el 9, una tecla para cambiar el signo, una tecla para borrar el último número que haya sido ingresado, una tecla llamada ENTER para continuar adelante o para realizar los cálculos y tres teclas para el ingreso de la función a calcular (Seno, Cose o Tangente) Ver Figura 1-1

Figura 1-1 Teclado hexadecimal 4x4 utilizado en la implementación de este proyecto

- Implementación de un IP Core para el cálculo de funciones Senos y Cosenos usando el Algoritmo CORDIC.
- El IP Core implementado calcula simultáneamente los valores de seno y coseno, de este modo, para obtener el valor de tangente no es necesario implementar un segundo IP Core, tan solo dividimos los valores obtenidos a la salida del IP Core.
- Uso del bus Avalon para interconectar el IP Core implementado con los demás bloques del sistema en chip.
- Cálculo de 3 funciones trascendentales: Seno, Coseno y Tangente.

- Los datos ingresados por el usuario y los resultados de las funciones calculadas serán visualizados en el módulo LCD 16x2 que incluye la tarjeta DE2 de Altera.

- Para el cálculo de Senos, Cosenos y Tangente, los valores de ángulos ingresados sólo pueden ser enteros positivos o negativos, con un máximo de 5 dígitos.

- Es posible borrar el valor de ángulo ingresado por el usuario hasta antes de realizar el cálculo de la función.

- El resultado de una función incluirá su signo correspondiente.

- Una vez que se obtiene un resultado, es posible volver a realizar un nuevo cálculo presionando la tecla ENTER.

- Las limitaciones de este proyecto son las siguientes:

- Solo se calcularán 3 funciones trascendentales Seno, Coseno y Tangente.

- No es posible ingresar valores decimales como parámetro de las funciones a calcular.

- Aunque la función seno y coseno se calculan simultáneamente con el IP Core implementado, en el display solo se mostrará el valor de resultado de la función escogida.

- El algoritmo de CORDIC nos devuelve un valor aproximado al resultado de una función.

- El resultado de la función calculada solo tendrá 4 dígitos decimales.

- No se implementará un control de interrupciones en nuestro sistema.

1.3 JUSTIFICACIÓN DEL PROYECTO.

En la actualidad ya existen dispositivos que hacen el cálculo de funciones trascendentales como por ejemplo, las calculadoras científicas; pero la finalidad de este proyecto es usar una tecnología moderna que nos permita crear un sistema en un chip programable (SOPC), que realice el cálculo de dichas operaciones y que permita realizar el cálculo de nuevas funciones, sin tener que aumentar su hardware, esta es la razón por la que usamos la tecnología de las FPGAs.

Sabemos que hoy en día, la tendencia es crear grandes sistemas con mayor eficiencia y con menos hardware posible, una buena opción para el

desarrollo de la máquina calculadora de este proyecto es usa el algoritmo CORDIC, que permite el cálculo de funciones complejas de una forma muy simple, rápida y con buena precisión (la precisión del algoritmo depende del número de iteraciones usadas para su cálculo), sin requerir hardware caro, como por ejemplo, una unidad de multiplicación, ya que este algoritmo se basa principalmente en sumas y desplazamientos.

Crear sistemas con poco hardware permite que los costos se vean reducidos y el consumo de energía sea menor.

Un indicador de eficiencia de un sistema, es su tiempo de respuesta, el algoritmo CORDIC por su sencillez es capaz de dar una respuesta en menor tiempo en comparación con otros algoritmos que hacen lo mismo.

Como podemos ver, la maquina calculadora de funciones trascendentales basada en algoritmo CORDIC, es un SOPC eficiente, no requiere de componentes complejos, y cuyo diseño puede ser modificado en cualquier momento para agregar o editar su funcionalidad sin alterar el hardware en el que se implementa.

CAPÍTULO 2

2 MARCO TEORICO

En este capítulo se explican los fundamentos teóricos de los Sistemas en Chips Configurables y también acerca del hardware y software que usaremos para la implementación de nuestro proyecto. Además se explica el algoritmo de CORDIC, su demostración matemática y sus ventajas en la parte final de este apartado.

2.1 SISTEMA EN CHIP CONFIGURABLE

SOC (System on a chip), como su propio nombre indica, integra en un solo chip los diferentes componentes de un sistema informático o electrónico, dando como resultado un sistema embebido en un chip que es capaz de realizar una función de manera un poco compleja pero más útil en comparación con otros sistemas no integrados. Así pues, en lugar de

construir un producto electrónico montando varios chips y componentes en una placa de circuito, la tecnología SOC permite que todas estas partes se fabriquen en un solo chip, y que el producto resultante cumpla con la misma funcionalidad. Por lo tanto, con menos piezas necesarias para el sistema, los costes de material y ensamblado se ven reducidos.[1]

Un SoC normalmente consume menos energía, tiene un coste inferior y una mayor fiabilidad que los sistemas multi-chip[2].

Un ámbito común de aplicación de la tecnología SoC son los **sistemas embebidos**.

Los SoCs pueden ser fabricados usando diferentes tecnologías como:

- Diseño a medida (Full Custom)
- Diseño basado en celdas estándares (standard cell)
- Diseño basado en FPGAs.

[2] Sistema multi-chip es un encapsulado especializado donde múltiples circuitos integrados (CIs), matrices de semiconductores u otros componentes discretos, son empaquetados en un substrato unificado, facilitando su uso como un solo componente (como si fuera un CI más grande).

2.1.1 COMPONENTES BÁSICOS DE UN SOC

Un SoC estándar está constituido por:

- Un microcontrolador, microprocesador o núcleo DSP.
- Módulos de memoria (informática) como ROM (memoria de sólo lectura), RAM (memoria de acceso aleatorio), EEPROM (memoria de sólo lectura programable y borrable electrónicamente) y Flash (memorias NAND de acceso muy rápido en comparación con los tradicionales -todavía hoy muy usados- soportes magnéticos).
- Generadores de frecuencia fija como por ejemplo osciladores y/o lazos de seguimiento de fase o PLLs.
- Componentes periféricos como contadores-temporizadores, temporizadores o relojes de tiempo real y generadores PoR[3]
- Interfaces externas incluyendo estándares como USB, IEEE 1394/Firewire, Ethernet, USART, o SPI.
- Interfaces analógicas que incluyen ADCs y DACs.
- Reguladores de voltaje y circuitos de Power Management.

[3] PoR: power-on reset, dispositivos que reajustan el sistema al recibir señal positiva, permitiendo a un sistema electrónico arrancar desde un estado conocido

- Estos módulos están interconectados de acuerdo a estándares industriales de conexión de buses, como también a tecnologías propietarias como por ejemplo la Especificación AMBA.[1]

2.1.2 SISTEMAS EMBEBIDOS

Un sistema embebido o empotrado es un sistema computacional[4] que posee hardware de computador y software embebido en un chip reprogramable, es diseñado para realizar una o unas pocas funciones específicas en tiempo real,

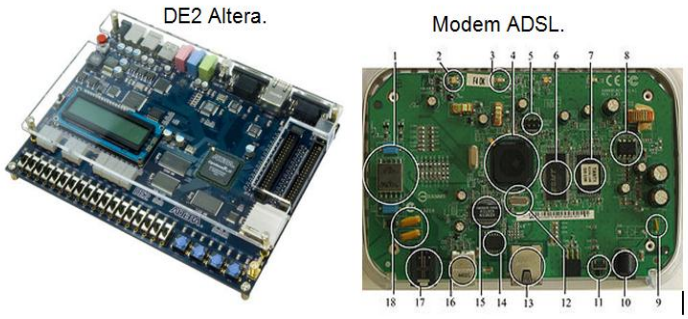


Figura 2-1 Sistemas embebidos.

La mayoría de los componentes de hardware se encuentran incluidos en su placa base (audio, puertos USB, botoneras, otros) y muchas veces los dispositivos resultantes de un sistema embebido no tienen el aspecto de

[4] Sistema Computacional: Es una máquina electrónica que recibe y procesa datos para convertirlos en información útil.

algo parecido a un computador. Un sistema embebido puede ser independiente o ser parte de un sistema mayor. Figura 2-1.

Los sistemas embebidos poseen ciertas características que los distinguen de otros sistemas de cómputo, las más sobresalientes son:

- Funcionamiento específico: Un sistema embebido usualmente ejecuta un programa específico de forma repetitiva.

- Fuertes limitaciones como, costo, tamaño, desempeño, y consumo de energía. Los sistemas embebidos generalmente deben ser poco costosos, poseer un tamaño reducido, tener un buen desempeño para procesar datos en tiempo real, y además consumir un mínimo de energía para extender el tiempo de vida de las baterías o prevenir la necesidad de elementos adicionales de enfriamiento.

- Reactivos y tiempo real: Muchos sistemas embebidos deben ser reactivos o reaccionar ante cambios en el ambiente, además de realizar algunos cálculos en tiempo real sin ningún retraso, es decir, se deben tener resultados en tiempos fijos ante cualquier eventualidad. Por ejemplo, el módulo de control de viaje de un automóvil continuamente monitorea la velocidad y los sensores de frenos, reaccionando ante cualquier eventualidad. Ante un estímulo

anormal, el módulo de control debe realizar los cálculos de forma precisa y acelerada para garantizar la entrega de los resultados dentro de un tiempo límite, una violación en este tiempo podría ocasionar la pérdida del control del automóvil.[2]

Algunos ejemplos de sistemas embebidos son taxímetros, lectores biométricos, sensores de temperatura, una máquina para vender minutos de operadoras celulares, controladores de lavadoras, sistemas de alarma, cámaras, fax, scanners, microondas, termostatos, GPS' otros. Se podría decir que cualquier dispositivo que funcione con electricidad ya tiene o tendrá un sistema embebido incluido.

2.2 HARDWARE Y SOFTWARE DE UN SISTEMA EMBEBIDO

Un sistema embebido tiene tres componentes principales:

- Hardware.
- Un software primario o aplicación principal, que lleva a cabo una tarea en particular, o en algunas ocasiones una serie de tareas.
- Un sistema operativo que permite supervisar la(s) aplicación(es) por parte del usuario, además de proveer los mecanismos para la ejecución de

procesos. En muchos sistemas embebidos es requerido que el sistema operativo posea características de tiempo real.[2]

En la actualidad el desarrollo de sistemas embebidos realizado en universidades y centros de investigación prefiere el uso de FPGAs, debido a la enorme ventaja que poseen al ser reprogramables, lo que añade flexibilidad al flujo del diseño y reducción en el tiempo de desarrollo.

Para la programación y procesamiento de la información utilizamos la tarjeta DE2 de Altera, la cual cuenta con un sin número de elementos, entre esos se encuentra el dispositivo FPGA Cyclone II que ofrece memorias embebidas, también da la posibilidad de generar el procesador embebido Nios II gracias a las herramientas de SOPC Builder en Quartus II y así permitirnos implementar una variedad de proyectos de diseño para el desarrollo de sistemas digitales sofisticados.

2.2.1 TARJETA DE2 DE ALTERA CON CHIP CYCLONE II

La tarjeta DE2 Altera FPGA Cyclone® II 2C35, es una tarjeta de desarrollo que fue creada en un tamaño compacto y está diseñada con un fin educativo, y ofrece un rico conjunto de características que la hacen ideal para el aprendizaje sobre lógica digital, organización de

computadoras, y FPGAs. Es adecuada para la implementación de aplicaciones simples que nos permiten entender los conceptos fundamentales para diseños avanzados.

El objetivo de la tarjeta es proporcionar el vehículo ideal para prototipos de diseño avanzado en almacenamiento, multimedia y redes.

Altera proporciona un conjunto de materiales de apoyo para la placa DE2, incluyendo tutoriales, ejercicios de laboratorio y demostraciones ilustrativas.

El kit de la Tarjeta de Desarrollo Altera DE2 posee:

- Una tarjeta DE2 de 8'' x 6'' con un FPGA Cyclone II EP2C35.
- Adaptador de 9V AC/DC
- Cable USB
- Cobertor para Tarjeta
- Guía de Instalación
- CD

Información de la Tarjeta DE2	
FPGA	• Cyclone II EP2C35F672C6 con EPCS16 16-Mbit • configuración serial del dispositivo
Interfaces E/S	• USB-Blaster para la configuración FPGA • Linea In/Out, Micrófono (24-bit Audio CODEC) • Video Out (VGA 10-bit DAC) • Video In (NTSC/PAL/Multi-format) • RS232 • Puerto Infrarrojo • PS/2 puerto para el mouse o el teclado • 10/100 Ethernet • USB 2.0 (tipo A y tipo B) • Cabecera de Expansión (Dos de 40 pines)
Memoria	• 8 MB SDRAM, 512 KB SRAM, 4 MB Flash • Puerto para la tarjeta SD
Displays	• Ocho displays de 7-segmentos • Display LCD de 16 x 2
Switches y LEDs	• 18 switches • 18 LEDs rojos • 9 LEDs verdes • 4 botoneras
Relojes	• 50 MHz clock • 27 MHz clock • Entrada SMA de clock

Tabla 2-1 Información de la Tarjeta DE2

2.2.2 FPGA

Un FPGA (Field-Programmable Gate Array) es un circuito integrado que puede configurarse para llevar a cabo cualquier función. Su arquitectura consiste en arreglos de varios bloques programables (bloques lógicos)

que mediante canales de conexiones verticales y horizontales se interconectan con celdas de entrada/salida y entre sí, tal como muestra la Figura 2-3.

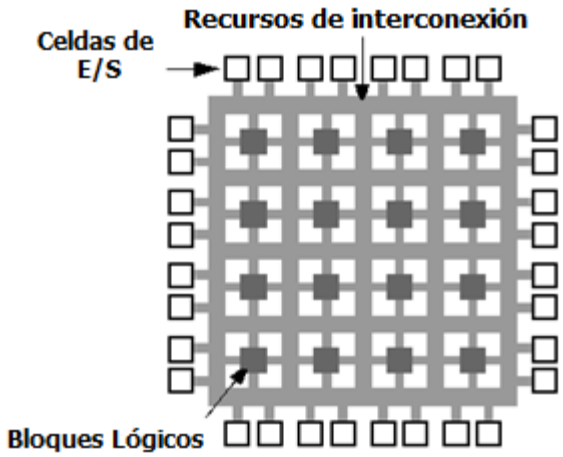


Figura 2-2 Arquitectura básica de un FPGA

Para programar un FPGA, el diseñador normalmente debe seguir la especificación de un lenguaje de descripción de hardware conocido como HDL. Ejemplos de tal lenguaje son: VHDL, Verilog.

La programación se logra con la ayuda de programas que brindan un entorno de desarrollo, como por ejemplo Max+Plus II y QuatusII.

La tarea del "programador" es definir la función lógica que realizará cada uno de los CLB, seleccionar el modo de trabajo de cada IOB e

interconectarlos, es decir, el "programador" indica al chip las operaciones que debe realizar con los datos que obtiene por medio de periféricos de entrada como puerto RS232, botoneras, y otros presentes en la tarjeta **DE2**, para una vez obtenidos los datos útiles (luego del respectivo proceso de datos) enviarlos a periféricos de salida como leds, display LCD, etc.

Para trabajar con un FPGA se debe contar con un software especial creado por el fabricante.

Existen varios fabricantes de chips y FPGAs enfocados al desarrollo de sistemas embebidos, entre ellos tenemos:

- Xilinx
- Altera
- Atmel
- Cypress
- Lattice Semiconductors
- Quicklogic

- Achronix Semiconductor

Siendo los dos primeros en la lista los más grandes productores.

En el desarrollo de nuestro proyecto utilizamos una tarjeta Altera DE2 que incluye un chip Cyclone® II 2C35 FPGA.

Para programar el chip utilizamos la herramienta de desarrollo Quartus II y el lenguaje de programación VHDL.

2.2.3 PROCESADOR EMBEBIDO NIOS II

El NIOS II es un núcleo procesador configurable que se puede implementar en alguna de las tres versiones disponibles, ya sea para minimizar el consumo de recursos de la FPGA o maximizar el rendimiento del procesador:

Nios II/f (fast): Es la versión diseñada de alto rendimiento y que con un pipeline[5] de 6 etapas proporciona opciones específicas para aumentar su desempeño, como memorias caché de instrucciones y datos.

[5] Pipeline es un conjunto de elementos procesadores de datos conectados en serie, en donde la salida de un elemento es la entrada del siguiente.

Nios II/S (standar): Es la versión con pipeline de 5 etapas, que dotada de unidad aritmético-lógica busca combinar rendimiento y consumo de recursos.

Nios II/e (economic): Es la versión que requiere menos recursos de la FPGA, sin pipeline y muy limitada, dado que carece de las operaciones de multiplicación y división.

Cada una de estas versiones se completa con una serie de componentes, memorias y periféricos, que se interconectan entre sí a través de un bus de sistema al que denominan “Avalon Switch Fabric” con la finalidad de obtener un sistema NIOS II completo en un chip (SoC).[4]

2.2.3.1 CARACTERÍSTICAS Y ARQUITECTURA

El NIOS II es un procesador de 32 bits de propósito general, basado en una arquitectura de tipo Harvard[6], dado que usa buses separados para instrucciones y datos; y cuyas principales características son:

- Tamaño de palabras de 32 bits.
- Juego de instrucciones RISC de 32 bits.

[6] La arquitectura Harvard cuenta con la memoria de programa y la memoria de datos separado y solo accesible a través de buses distintos.

- 32 registros de propósito general de 32 bits cada uno (del r0 al r31).
- 6 registros de control de 32 bits (del ctl0 al ctl5).
- 32 fuentes de interrupción externa.
- Capacidad de direccionamiento de 32 bits.
- Operaciones de multiplicación y división de 32 bits.
- Acceso a los periféricos integrados e interfaces.

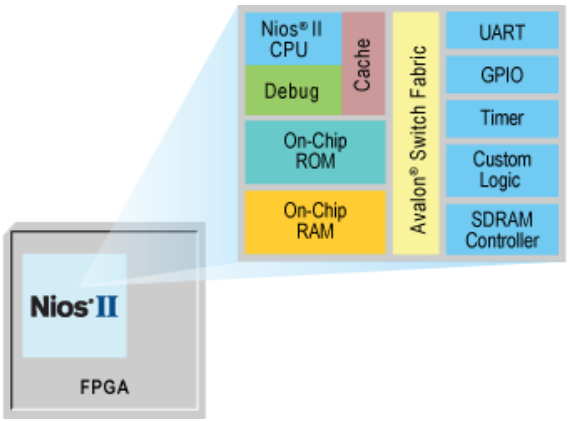


Figura 2-3 Sistema basado en NIOS II

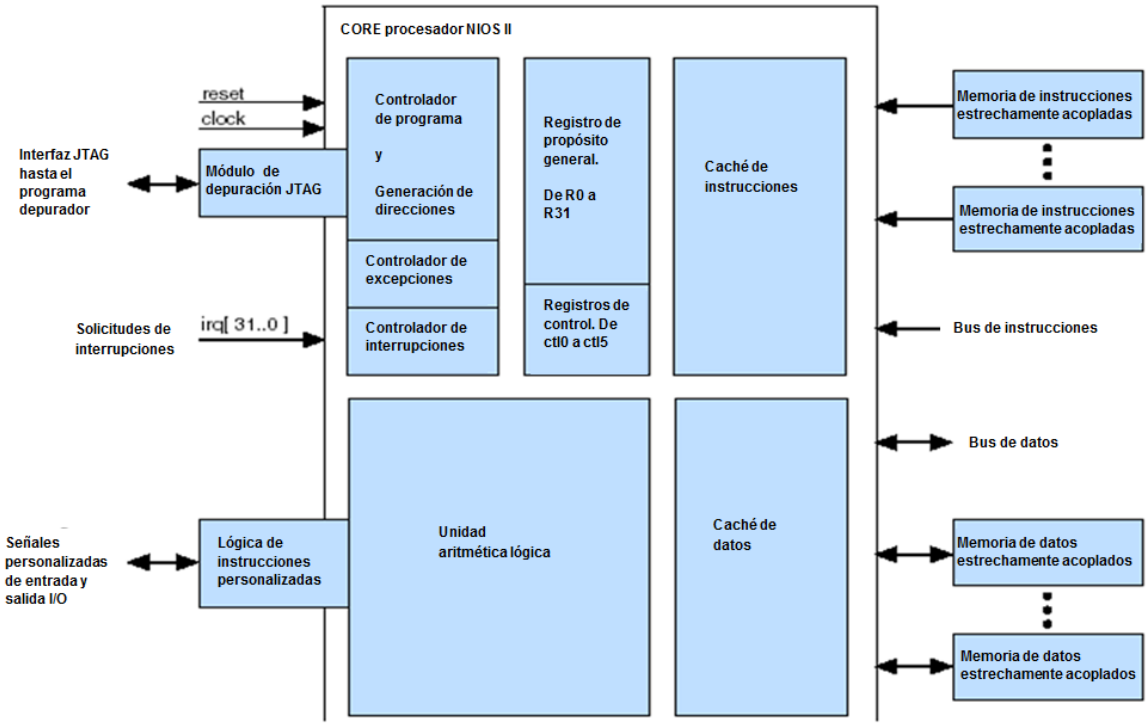


Figura 2-4 Diagrama Bloques que define la arquitectura del NIOS II.

Como se observa en la Figura 2-4 el procesador NIOS II está formado por una serie de unidades funcionales, que están dadas según la versión implementada. Entre estas, las principales son: los registros, la unidad aritmético-lógica, la interfaz para instrucciones definidas por el usuario, el controlador de excepciones, el bus de instrucciones, el bus de datos, la memoria caché de instrucciones y datos, la interfaz de acoplamiento directo de memoria de instrucciones y datos y el módulo de depuración JTAG.[4]

2.2.3.2 GESTIÓN DE EXCEPCIONES

El controlador de excepciones es el circuito encargado de realizar las tareas ligadas a la atención de excepciones, que son aquellas situaciones anormales que interrumpen el flujo de ejecución de un programa, y que se requiere la atención del procesador.

Cuando se produce una excepción, el procesador transfiere la ejecución al controlador de excepciones, que en este caso, las gestiona a través de una dirección de memoria.

El Nios II incluye en todas sus versiones un controlador interno de interrupciones (ICC) y ofrece la posibilidad de configurar el bloque controlador de interrupciones externo (EIC).[4]

2.2.3.3 CONEXIÓN Y ACCESO A MEMORIA

El Nios II utiliza 32 bits para realizar un direccionamiento por byte, poseyendo un bus para instrucciones y otro para datos, que corresponde a una arquitectura tipo Harvard.

En este tipo de conexión, el bus de instrucciones se utiliza para leer el programa situado en memoria, que debe ser ejecutado por el

procesador, mientras que el bus de datos proporciona acceso a los diferentes bloques de memoria y a los periféricos del sistema. Sin embargo, existen tres modos posibles de conexión con memoria:

- Conexión de memoria a través del bus Avalon, tanto de memoria interna como externa.
- Conexión directa de bloques de memoria interna, con lo que se proporciona un rápido acceso a memoria.
- Conexión de memoria a través de caché.

2.2.3.4 CONEXIÓN Y ACCESO A PERIFÉRICOS

En el NIOS II, la memoria de datos y los periféricos comparten la capacidad de direccionamiento de 32 bits, y por tanto el bus. Así, el acceso de entrada/salida (I/O) se realiza a través del mapa de memoria, del mismo modo que cualquier otra dirección de memoria.

2.2.3.5 BUS AVALON

Está diseñado para la interconexión de los diferentes componentes o bloques de diseño que conforman un sistema, se puede identificar tradicionalmente, cuatro arquitecturas de conexión diferentes:

Conexión en bus: Se basa en la conexión de maestros y esclavos por medio de una unidad de arbitraje común, lo que permite operar a frecuencias relativamente altas a costa de perder la concurrencia.

Conexión full crossbar switch: Se basa en la conexión directa de maestros y esclavos por medio de una matriz de conexiones, lo que permite transacciones concurrentes entre los diferentes elementos del sistema, siendo uno de sus usos principales las aplicaciones de computación de alto rendimiento.

Conexión partial crossbar switch: se basa en una conexión directa de un maestro con un determinado grupo de esclavos, lo que permite ahorrar recursos en la creación de la matriz de conexiones.

Conexión en streaming: se basa en la conexión directa entre fuente (source) y sumidero (sink), creando un flujo de datos unidireccional para la transferencia de datos a alta velocidad, dado que elimina la

unidad de arbitraje al crear una conexión punto a punto, siendo uno de sus principales usos el procesamiento de video.

Altera, basándose en una arquitectura “Partial Crossbar Switch”, implementa una serie de interfaces a las que designa como AVALON, que facilitan la interconexión de componentes en sistemas complejos y permiten la interconexión de sistemas concurrentes.

2.2.3.6 CONJUNTO DE INSTRUCCIONES

El NIOS II presenta un conjunto de instrucciones RISC, de 32 bits de longitud. Además, su lenguaje ensamblador soporta una serie de pseudoinstrucciones que se traducen en varias instrucciones en lenguaje máquina para agilizar el desarrollo de programas a bajo nivel. Las instrucciones son de tipo: carga y almacenamiento, aritméticas, lógicas, de transferencia entre registros, de comparación, de rotación y desplazamiento, de ruptura y salto, de llamada a subrutinas, manejo de excepciones e instrucciones de control.

2.2.4 QUARTUS II

Quartus II es una herramienta avanzada de diseño que la compañía ALTERA ha creado para diseñar sistemas digitales e implementarlos en dispositivos de lógica programable

Este software ofrece un completo entorno de desarrollo multiplataforma que se adapta fácilmente a las necesidades específicas de diseño de SOPC. Esta herramienta permite el análisis y la síntesis de diseños realizados en lenguaje HDL, examinar diagramas RTL.

Quartus II incluye soluciones para todas las fases de diseño de FPGA y CPLD. [6]

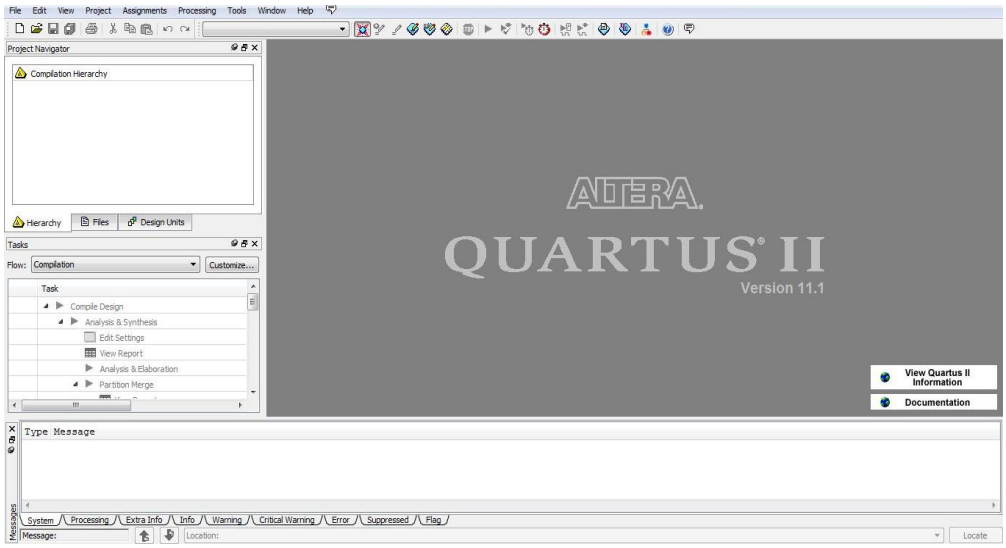


Figura 2-5 Ventana principal de Quartus II

2.2.5 SOPC BUILDER

Es una herramienta de desarrollo de sistema, que permite definir y generar un completo SOPC de alto rendimiento en mucho menos tiempo que el método tradicional. SOPC Builder se incluye como parte del software Quartus II.

Esta herramienta permite diseñar la estructura de un sistema de hardware, que por medio de una interfaz gráfica de usuario, permite agregar componentes a un sistema, configurar los componentes, y especificar la conectividad.

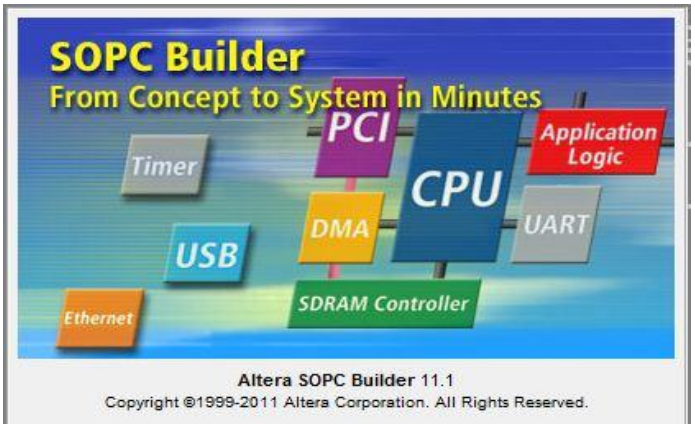


Figura 2-6 Ventana de inicio de SOPC Builder

El constructor SOPC, automatiza la tarea de integrar componentes de hardware. En el uso de métodos tradicionales de diseño, se debe escribir manualmente los módulos de HDL para cablear juntos las piezas del

sistema. Mediante el Generador de SOPC, se especifican los componentes del sistema en una interfaz gráfica de usuario y el constructor SOPC genera la lógica de interconexión de forma automática. SOPC Builder genera archivos de HDL que definen todos los componentes del sistema y un archivo de HDL de nivel superior que conecta todos los componentes juntos.

Sin embargo, SOPC es una herramienta de propósito general para la creación de sistemas que pueden o no pueden contener un procesador y puede incluir un procesador suave que no sea el procesador Nios II.

2.2.6 NIOS II SBT PARA ECLIPSE

Previo a explicar sobre esta herramienta de software, es necesario entender que Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma que permite desarrollar "Aplicaciones de Cliente Enriquecido"7. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que son parte de Eclipse.

[7] El objetivo del cliente enriquecido consiste en proporcionar una interfaz gráfica, escrita con una sintaxis basada en XML, que proporciona funcionalidades similares a las del cliente pesado (arrastrar y soltar, pestañas, ventanas múltiples, menús desplegables).

Para programar el procesador Nios II se utiliza NIOS II SBT para Eclipse, que es una plataforma de desarrollo de software para el NIOS II.

El Nios II SBT para Eclipse es un conjunto de plugins basados en la infraestructura de Eclipse, nos permite crear interfaces usuario-hardware que se pueden escribir en varios lenguajes de programación, por ejemplo: Lenguaje Assembler, C, C++, Java.

En nuestro caso, la herramienta de software que usamos para el desarrollo del sistema operativo es Nios II para Eclipse, y para la programación empleamos el lenguaje C.

Esta herramienta puede llevar a cabo la mayoría de tareas de desarrollo de software dentro de Eclipse, incluyendo la creación, edición, creación, ejecución, depuración y perfilado de los programas.

Figura 2-7 Ventana de inicio de Eclipse

2.3 EL ALGORITMO CORDIC

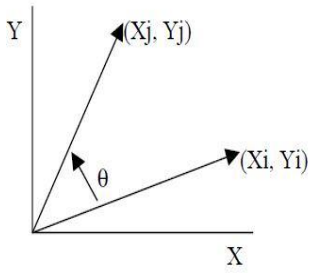
CORDIC (Coordinate Rotation Digital Computer) es un algoritmo originalmente propuesto por Jack Volder en el año 1959 y es un método para calcular funciones elementales usando lo mínimo en hardware, como registros de desplazamientos, sumadores/restadores y comparadores.

CORDIC trabaja rotando el sistema de coordenadas a través de ángulos constantes hasta que un ángulo sea reducido a cero. Los desplazamientos de ángulos son seleccionados de tal manera que las operaciones en los X y Y son solo desplazamientos y sumas.

La rotación de vectores puede utilizarse a su vez para conversión de sistemas de coordenadas (cartesiano a polar y viceversa), magnitud de vectores y como parte de funciones matemáticas más complejas como la Transformada Rápida de Fourier (FFT) y la Transformada Coseno Discreta (DCT).

2.3.1 FUNDAMENTO TEÓRICO

El algoritmo CORDIC realiza una rotación plana. Gráficamente, una rotación plana significa transformar un vector (Xi, Yi) en un nuevo vector (Xj, Yj).



Usando una forma matricial, una rotación planar para un vector (Xi, Yi) está definida como:

$$\begin{bmatrix} X_j \\ Y_j \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \end{bmatrix} \qquad (2.1)$$

El ángulo de rotación teta (θ) puede ser ejecutado en varios pasos, usando un proceso iterativo. Cada paso realiza una pequeña parte de la rotación, muchos pasos compondrán una rotación completa. Un paso pequeño en la rotación está definido como:

$$\begin{bmatrix} X_{n+1} \\ Y_{n+1} \end{bmatrix} = \begin{bmatrix} \cos\theta_n & -\sin\theta_n \\ \sin\theta_n & \cos\theta_n \end{bmatrix} \begin{bmatrix} X_n \\ Y_n \end{bmatrix} \tag{2.2}$$

La ecuación 2.3 puede ser modificada eliminando el factor $\cos\theta_n$.

$$\begin{bmatrix} X_{n+1} \\ Y_{n+1} \end{bmatrix} = \cos\theta_n \begin{bmatrix} 1 & -\tan\theta_n \\ \tan\theta_n & 1 \end{bmatrix} \begin{bmatrix} X_n \\ Y_n \end{bmatrix} \tag{2.3}$$

La ecuación 2.3 requiere tres múltiplos, en comparación con la ecuación 2.2 que solo necesita 2.

Multiplicadores adicionales pueden ser eliminados mediante la selección de un ángulo de paso cuya tangente es una potencia de 2. Multiplicar o dividir por una potencia de 2 se puede implementar utilizando una operación de desplazamiento simple.

El ángulo para cada paso está dado por:

$$\theta_n = \arctan\left(\frac{1}{2^n}\right) \tag{2.4}$$

La suma de todos los ángulos de paso debe ser igual al ángulo de rotación θ.

$$\sum_{n=0}^{\infty} S_n \theta_n = \theta \tag{2.5}$$

donde

$$S_n = \{-1;+1\} \tag{2.6}$$

Esto resulta en la siguiente ecuación para $\tan \theta_n$

$$\tan \theta_n = S_n 2^{-n} \tag{2.7}$$

Combinando la ecuación 2.3 y 2.7 obtenemos:

$$\begin{bmatrix} X_{n+1} \\ Y_{n+1} \end{bmatrix} = \cos \theta_n \begin{bmatrix} 1 & -S_n 2^{-n} \\ S_n 2^{-n} & 1 \end{bmatrix} \begin{bmatrix} X_n \\ Y_n \end{bmatrix} \tag{2.8}$$

Sin tomar en cuando el valor del coeficiente $\cos \theta_n$, el algoritmo ha sido reducido a unos cuantos desplazamientos y sumas simples. El coeficiente puede ser eliminado mediante el pre-cálculo de su resultado final. El primer paso es rescribir el coeficiente:

$$\cos\theta_n = \cos\left(\arctan\left(\frac{1}{2^n}\right)\right) \tag{2.9}$$

El Segundo paso es calcular la ecuación 2.9 para todos los valores de ‘n’ y luego multiplicando los resultados, obtendremos un resultado al cual nos referiremos como K.

$$K = \frac{1}{P} = \prod_{n=0}^{\infty} \cos\left(\arctan\left(\frac{1}{2^n}\right)\right) \approx 0.607253 \tag{2.10}$$

K (congregate constant) es una constante para todos los vectores iniciales y para todos los valores de los ángulos de rotación. Su derivada P (aprox. 1.64676) está definida aquí porque también es comúnmente usada.

Ahora podemos formular el cálculo exacto que CORDIC realiza.

$$\begin{cases} X_j = K\left(X_i \cos\theta - Y_i \sin\theta\right) \\ Y_j = K\left(Y_i \cos\theta + X_i \sin\theta\right) \end{cases}$$

(2.11)

A causa de que el coeficiente K ya ha sido pre-calculado para un largo número de pasos, la ecuación 2.8 podría ser escrita como:

$$\begin{bmatrix} X_{n+1} \\ Y_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & -S_n 2^{-n} \\ S_n 2^{-n} & 1 \end{bmatrix} \begin{bmatrix} X_n \\ Y_n \end{bmatrix} \tag{2.12}$$

O también como:

$$\begin{cases} X_{n+1} = X_n - S_n 2^{-2n} Y_n \\ Y_{n+1} = Y_n + S_n 2^{-2n} X_n \end{cases} \tag{2.13}$$

En este punto una nueva variable llamada Z es introducida. Z representa la parte del ángulo inicial que no ha sido rotada aún.

$$Z_{n+1} = \theta - \sum_{i=0}^{n} \theta_i \tag{2.14}$$

Para cada paso de la rotación Sn es calculado como un signo de Zn

$$S_n = \begin{cases} -1 & if \ \ Z_n < 0 \\ +1 & if \ \ Z_n \geq 0 \end{cases} \tag{2.15}$$

Combinando las ecuaciones 2.5 y 2.15, obtenemos un sistema el cual reduce la parte no rotada del ángulo θ a cero.

Tal como se muestra en las líneas del siguiente programa:

```
For n=0 to [inf]
    If (Z(n) >= 0) then
```

```
            Z(n + 1) := Z(n) – atan(1/2^n);
        Else
            Z(n + 1) := Z(n) + atan(1/2^n);
        End if;
    End for;
```

El valor de atan(1/2^i) está ya pre-calculado y almacenado en una tabla. [inf] es reemplazado con el número de iteraciones requeridas, las cuales son alrededor de 1 iteración por bit (16 iteraciones darán un resultado de16 bits).

Si nosotros agregamos el cálculo para los valores de X y Y, podremos completar un programa diseñado de acuerdo al algoritmo CORDIC.

```
    For n=0 to [inf]
        If (Z(n) >= 0) then
            X(n + 1) := X(n) – (Yn/2^n);
            Y(n + 1) := Y(n) + (Xn/2^n);
            Z(n + 1) := Z(n) – atan(1/2^n);
        Else
            X(n + 1) := X(n) + (Yn/2^n);
            Y(n + 1) := Y(n) – (Xn/2^n);
            Z(n + 1) := Z(n) + atan(1/2^n);
        End if;
    End for;
```

Este algoritmo es conocido como “llevar Z a cero” o “CORDIC Rotation”, debido a que el algoritmo empieza a reducir un ángulo inicial hasta el valor de cero, acumulando los valores de X y Y en cada paso de la rotación, que al final serán los valores correspondientes al seno y coseno de dicho ángulo.

El core de CORDIC calcula:

$$[X_j , Y_j , Z_j]=[P(X_i\cos(Z_i)-Y_i\text{sen}(Z_i)), P(Y_i\cos(Z_i)+\text{sen}(Z_i)),0] \quad (2.16)$$

Hay un caso especial para el método “reducir Z a cero”:

$$X_i=1/P =K =0.60725$$

$$Y_i=0$$

$$Z_i=\theta$$

$$[X_j , Y_j , Z_j]=[\cos\theta,\text{sen}\theta,0] \quad (2.17)$$

Otro escenario que es posible, es “reducir Y a cero” o “CORDIC vectoring”. El core de CORDIC calcula:

$$[X_j , Y_j , Z_j]=[P\sqrt{{X_i}^2 + {Y_i}^2}], 0, Z_i + \text{atan}(\frac{Y_i}{X_i})] \quad (2.18)$$

Para este escenario hay dos casos especiales:

Primer Caso

$X_i=X$

$Y_i=Y$

$Z_i=0$

$$[X_j , Y_j , Z_j]= [P\sqrt{{X_i}^2 + {Y_i}^2} , 0, Z_i + \text{atan} (\frac{Y_i}{X_i})] \quad (2.19)$$

$X_i=1$

$Y_i=a$

$Z_i=0$

$$[X_j , Y_j , Z_j]= [P\sqrt{1 + a^2}, 0, \text{atan} (a)] \quad (2.20)$$

En el modo de funcionamiento CORDIC rotación (rotation) se rota el vector de entrada un ángulo específico que se introduce como parámetro. El segundo modo, denominado vectorización (vectoring), rota el vector de entrada hacia el eje X, acumulando el ángulo necesario para efectuar dicha rotación

2.3.2 ARQUITECTURAS DE IMPLEMENTACIÓN DEL ALGORITMO CORDIC

- La arquitectura Bit-Paralela Desplegada
- La arquitectura Bit-Serie Iterativa
- La arquitectura Bit-Paralela Iterativa

Para este proyecto se implementó la arquitectura Bit-Paralela Desplegada.

2.4.2.1 La arquitectura Bit-Paralela Desplegada

La denominación de paralela se debe a la forma en que se opera con las componentes X, Y y Z. En esta arquitectura el diseño puede desplegarse [7][8] como muestra la Figura 2-8. El diseño se separa en etapas correspondientes a cada iteración. Cada etapa está compuesta por los mismos componentes, dos unidades de desplazamiento y dos sumadores algebraicos. Por consiguiente la salida de una etapa corresponde a la entrada de la siguiente etapa. Los valores iniciales para X_0, Y_0 y Z_0 se ingresan en paralelo a la primera etapa.

Este diseño introduce dos ventajas importantes. Las unidades de desplazamiento así como las constantes correspondientes a cada iteración pueden ser cableadas. La segunda ventaja radica en que el circuito es puramente combinatorio, y por lo tanto no se necesita de una unidad de control, simplificando enormemente el diseño. La desventaja que presenta esta arquitectura es la enorme cantidad de espacio que requiere su implementación. En el esquema se supone que las componentes tienen un ancho de m bits. La señal Modo indica el modo de Operación (Rotación o Vectorización).

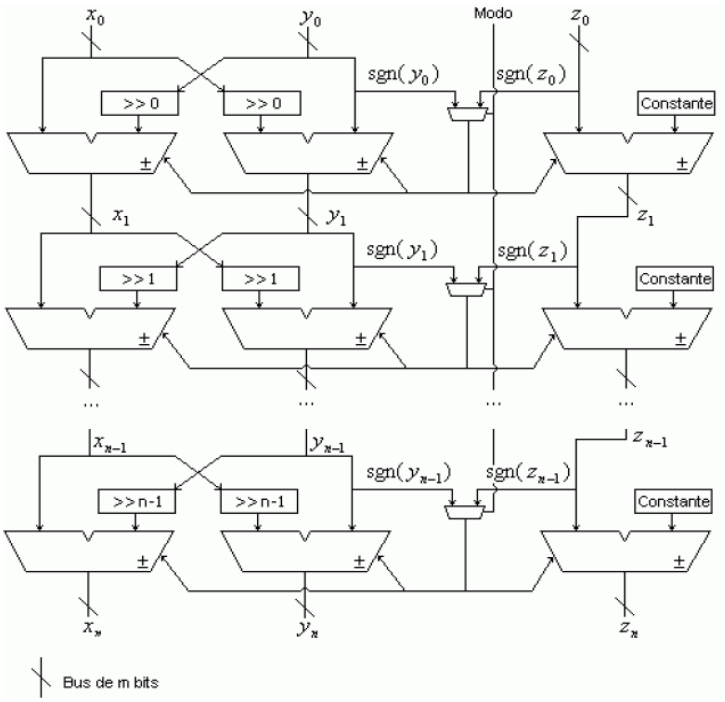


Figura 2-8 Esquema de la arquitectura Bit-Paralela Desplegada

2.3.3 VENTAJAS Y USOS DEL ALGORITMO CORDIC

Debido a que dicho algoritmo funciona en base a sumas, restas y desplazamientos, el hardware requerido para su implementación no requiere de unidades aritméticas sofisticadas y debido a esto consume menos energía en comparación con un sistema cuyo hardware trabaje realizando cálculos en su ALU.

Esto disminuye en gran parte el costo de inversión de hardware para desarrollo de proyectos donde se requiera el cálculo de funciones trascendentales que pueden obtenerse con el algoritmo CORDIC. Varios ejemplos del uso de este algoritmo son:

- Cálculo de matrices.
- El cálculo de transformadas discretas sinusoidales como la DFT, transformada discreta de Hartley, transformada discreta del Coseno (DCT), transformada discreta de Seno (DST) y la transformada Z, aplicadas en procesamiento de señales y aplicaciones de procesamiento de imágenes.
- Muchas aplicaciones de CORDIC en las comunicaciones usan el sistema de coordenadas circulares en uno o ambos modo de

operación de CORDIC. El modo “rotation”, es comúnmente usado para generar mezclas señales mezcladas, mientras que el modo “vectoring” es usado para estimar parámetros de fase y frecuencia.

- Dos de los problemas claves donde CORDIC provee área y soluciones en eficiencia de energía son:
  - ✓ Cinemática directa (DKS).
  - ✓ Cinemática inversa (IKS) de robots manipuladores.

CAPÍTULO 3

3 DISEÑO E IMPLEMENTACIÓN

En este capítulo se detalla el diseño e implementación del proyecto. A continuación detallamos las herramientas de hardware y software necesarias:

- Kit Tarjeta DE2 de Altera con chip Cyclone II
- 1 bus de datos
- 1 teclado hexadecimal 4x4
- 8 resistencias
- 1 PC con Quartus II 11.1, Eclipse, Nios II 11.1 Software Build tools for Eclipse, instalados respectivamente.

3.1 PRELIMINARES y JUSTIFICACIONES DEL DISEÑO

Para empezar, debemos imaginar el funcionamiento de una calculadora de bolsillo:

- Si queremos realizar una suma: ingresamos el primer dato, luego el signo de adición, seguido del siguiente número y por último el signo “igual”, con lo que obtendremos el resultado correspondiente.

- Si queremos obtener el valor de una función trigonométrica: presionamos la función a calcular, luego un valor de ángulo seguido del signo igual y obtenemos el resultado.

- Luego de citar los ejemplos anteriores podemos darnos cuenta que debemos:

- Ingresar datos de alguna manera y leerlos

- Seguir una secuencia en el ingreso de datos, signos o funciones

- Identificar la función a calcular, y realizar el cálculo respectivo

- Verificar que la secuencia de datos ingresados para el cálculo de una función sea correcta

- Mostrar los datos y resultados al usuario de alguna manera

- Contar con una interfaz para usuario.

Y como nuestra calculadora debe realizar cálculos de funciones senos y cosenos decidimos que su funcionamiento sea de la siguiente manera:

- Se inicia la calculadora y se debe ingresar una función a calcular (Seno, Coseno o Tangente)
- Se ingresa un valor de ángulo
- Se presiona la tecla ENTER
- Se realiza el cálculo respectivo
- Se muestran el resultado en pantalla y
- luego de unos segundos, se regresa al inicio del programa donde se debe elegir la función a calcular.

El ciclo de funcionamiento escogido para muestra calculadora se muestra en el siguiente diagrama de flujo.

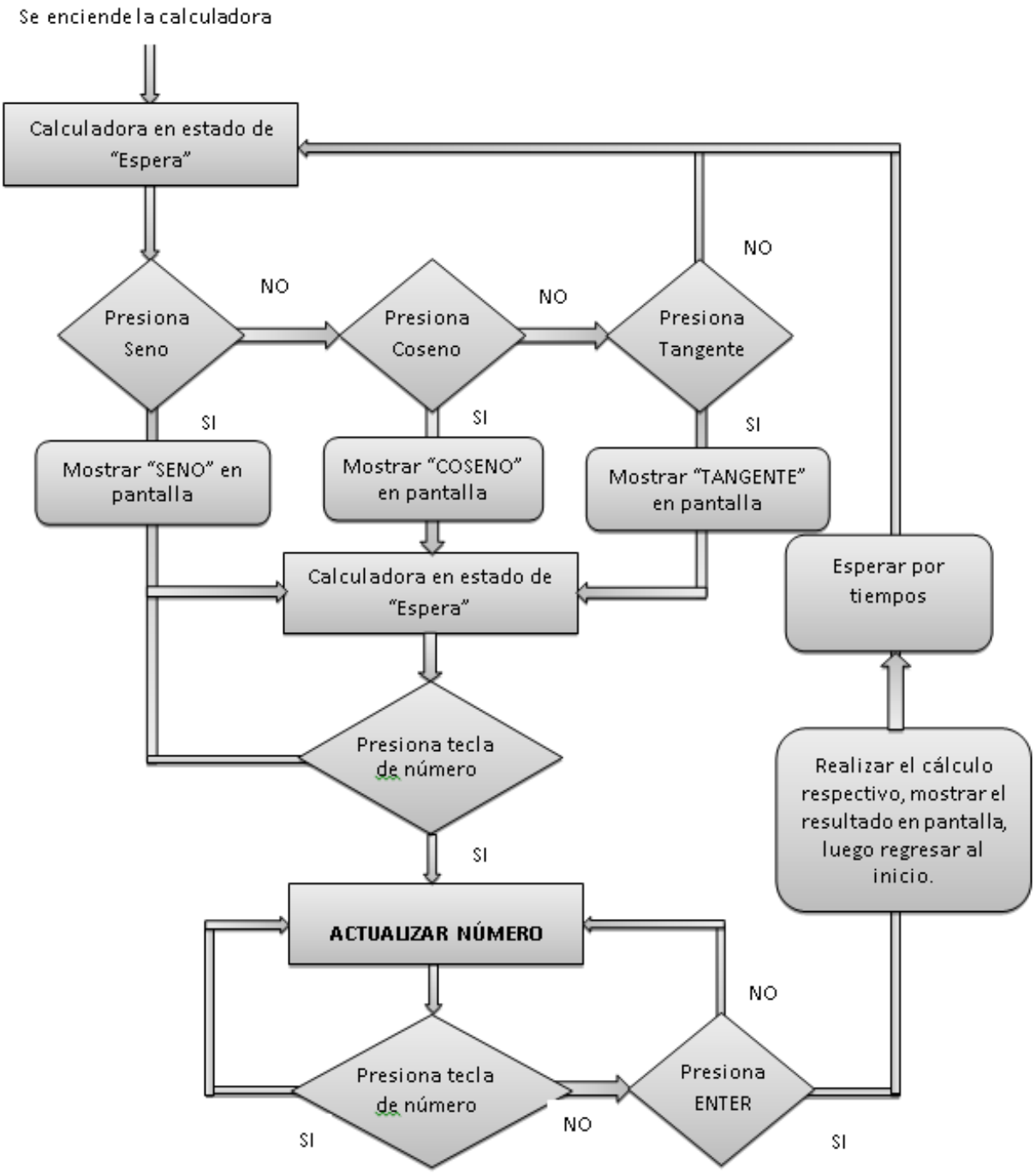


Figura 3-1 Diagrama de flujo que describe el funcionamiento de nuestra calculadora

3.1.1 INGRESO DE DATOS

Para el ingreso de datos, se decidió hacerlo a través de un teclado hexadecimal conectado al puerto de expansión JP2 incluido en la tarjeta DE2 (Figura. 3-2). Para ingresar los datos pudimos haber escogido usar los 18 switches que incluye la tarjeta y evitar conectar un teclado, sin embargo al usar los switches, el ingreso de un número debería hacerse con su correspondiente representación binaria, a diferencia de un teclado hexadecimal, que el ingreso lo hace con números del 0 al 9.

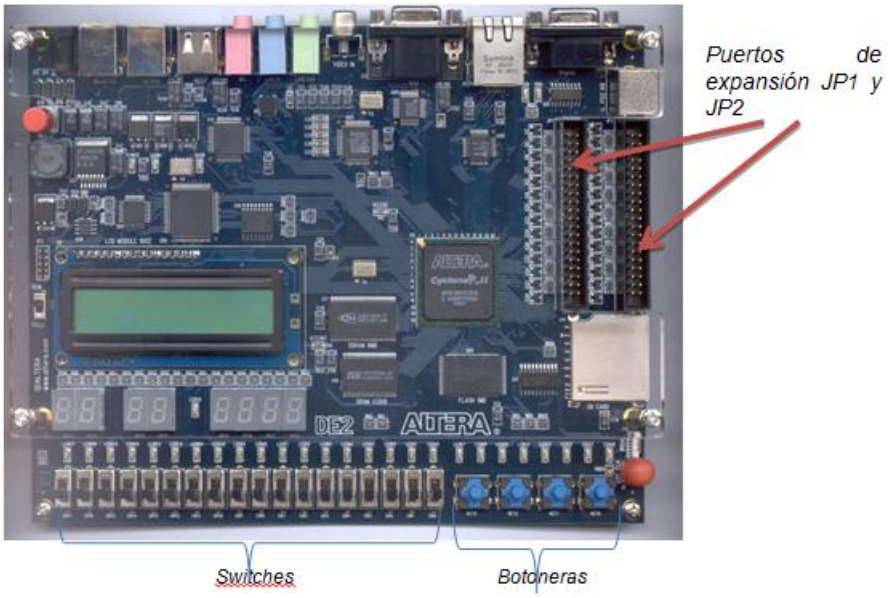


Figura 3-2 Algunos módulos para el ingreso de datos que incluye la tarjeta DE2

Para facilidad del usuario, en el ingreso de los datos usamos un teclado hexadecimal (con 16 teclas disponibles) que lo hemos adaptado de la siguiente manera: una parte numérica con teclas de 0 a 9, las teclas que indican las funciones trascendentales (SEN, COS, TAG) que desee calcular, la tecla ENT para indicar al sistema que se ha terminado una cadena de caracteres de entrada, la tecla DEL que es para borrar algún dato, y por último la tecla +/- que permite cambiar el signo del valor ingresado. Figura 3-3.

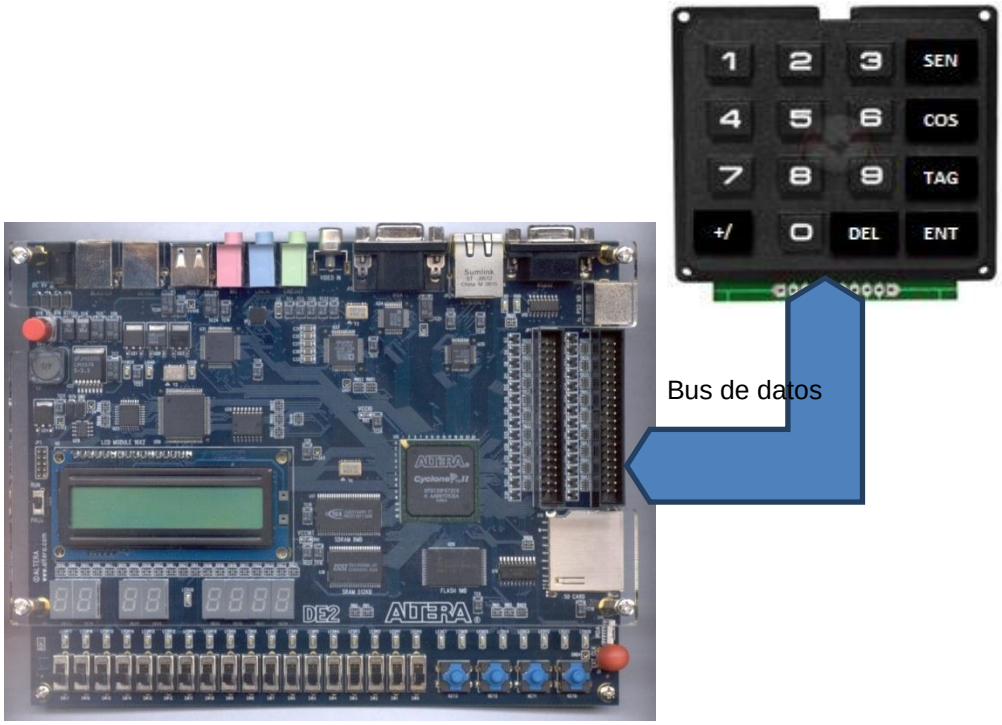


Figura 3-3 Teclado hexadecimal conectado a la tarjeta DE en el puerto JP2 por medio de un bus de datos.

3.1.2 MOSTRAR DATOS INGRESADOS Y RESULTADOS

En este proyecto utilizamos el display LCD 16x2 para mostrar al usuario los datos que ingresa por teclado y los resultados obtenidos de alguna función.

3.2 MODULOS DE LA TARJETA DE2 A USAR

A continuación se mencionan los módulos de la tarjeta DE2 que usamos en este proyecto:

- Chip Cyclone II
- Puerto de Expansion JP2 (conexión con teclado)
- LCD Display 16x2 (mostrar datos al usuario)
- Leds verdes (indicadores)

3.3 CREACIÓN DEL PROYECTO EN QUARTUS II

La sección de hardware de nuestro sistema basado en NIOS II es desarrollada en Quartus II, por lo que iniciamos por crear un nuevo proyecto en este software.

Ejecutamos Quartus II, creamos un nuevo proyecto utilizando el asistente.

File-> New Proyect Wizard…

Dentro de la primera ventana del asistente debemos definir un nombre para el proyecto y también el directorio de trabajo dentro del cual se guardarán todos los archivos concernientes al proyecto.

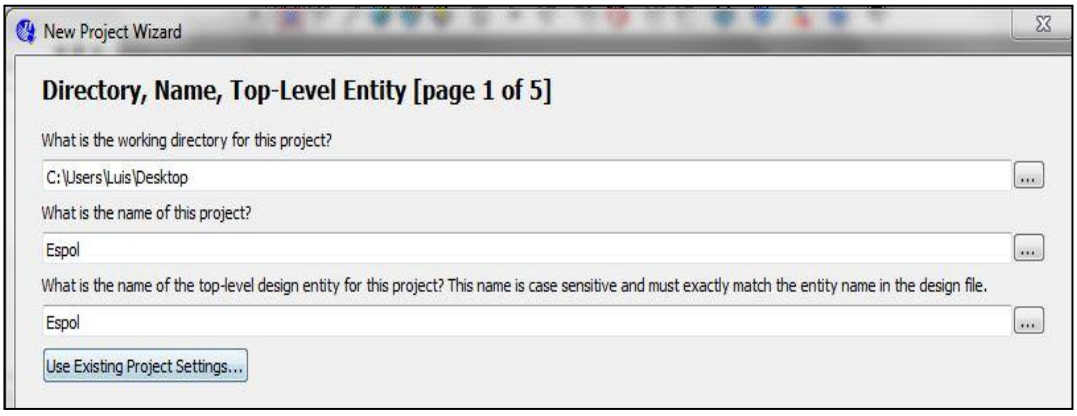


Figura 3-4 Ventana para definir el nombre a un nuevo proyecto.

En la siguiente ventana del asistente, se nos permite ingresar archivos existentes al proyecto, por ejemplo algún archivo VHDL que necesitemos usar luego.

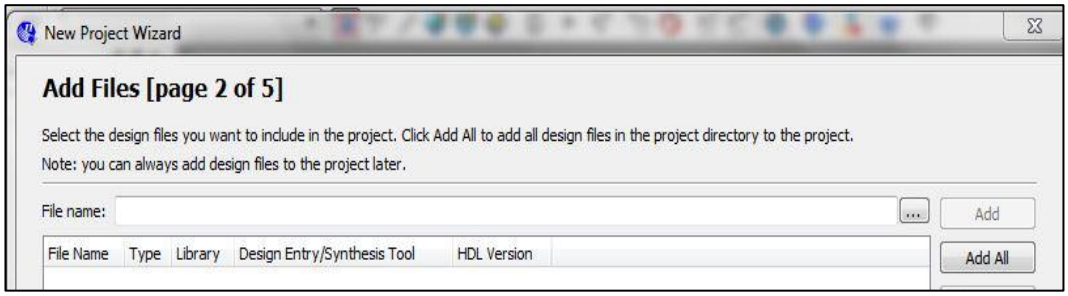


Figura 3-5 Ventana para ingresar archivos existentes al proyecto.

En la ventana 3 del asistente (Figura3-6), debemos indicar al programa cual es el dispositivo al cual cargaremos el proyecto a compilar. En nuestro caso ya que trabajamos con la tarjeta DE2 con chip Cyclone II, ingresamos los parámetros: Family->Cyclone II y device->EP2C35F672C6.

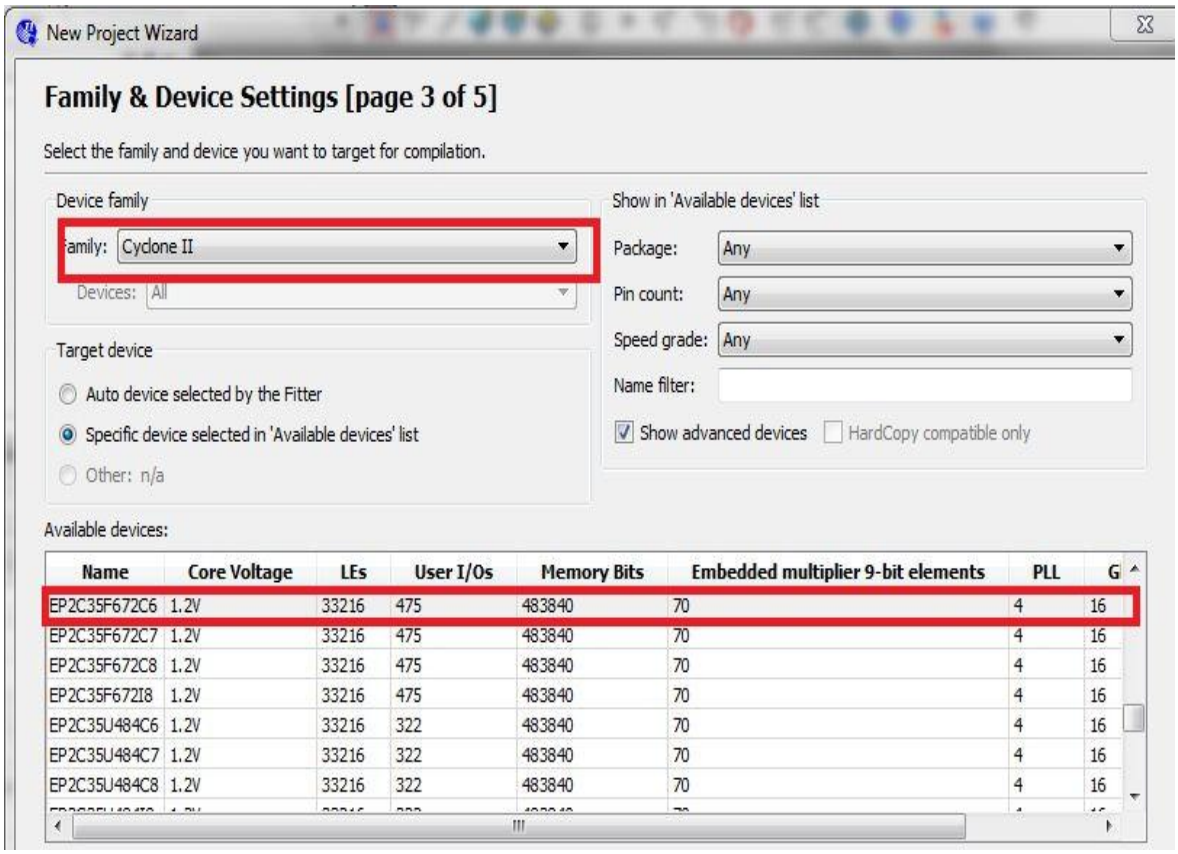


Figura 3-6 Ventana que permite indicar al programa el dispositivo al cual cargaremos el proyecto a compilar.

En el cuarto diálogo nos permite seleccionar diferentes herramientas para la compilación y simulación del sistema. Este tema se encuentra fuera del alcance de este documento por lo que se utilizaran las herramientas estándares.

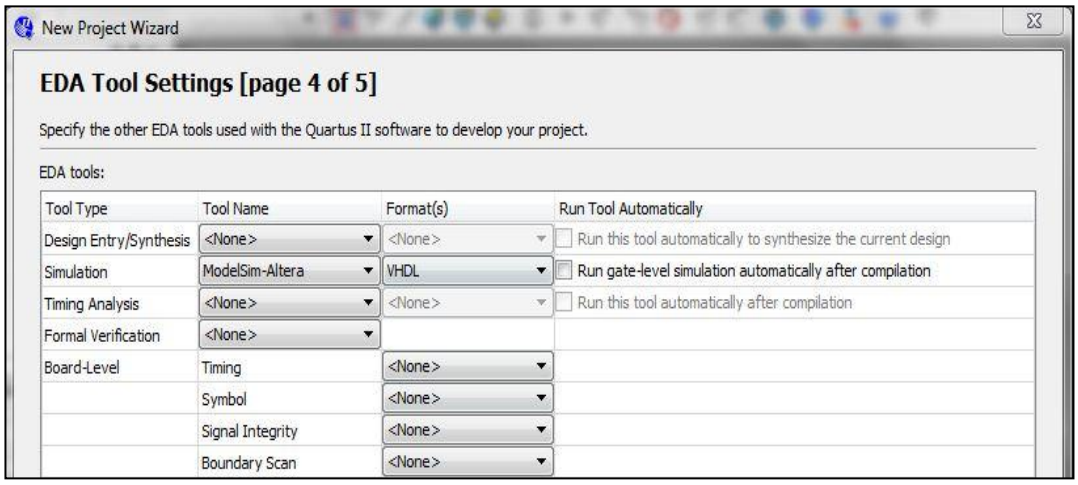


Figura 3-7 Ventana donde podremos seleccionar las herramientas para la compilación y simulación del sistema.

El último paso para crear un nuevo proyecto es finalizar el asistente.

La primera sección relacionada con el desarrollo del sistema de este proyecto dentro de Quartus II finaliza con la creación de un archivo esquemático donde se realizarán las conexiones del sistema NIOS II con los pines físicos del dispositivo reconfigurable.

File-> New -> Block Diagrama/Schematic File

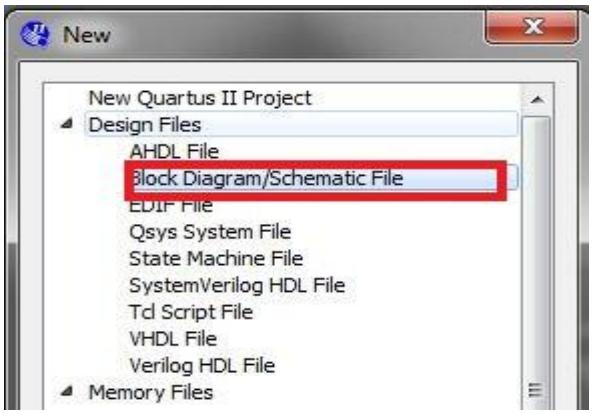


Figura 3-8 Ventana para la creación de un archivo esquemático.

3.4 DISEÑO DEL SISTEMA EN SOPC BUILDER

Sin cerrar la aplicación y manteniendo las configuraciones que hemos realizado, procedemos a trabajar en SOPC Builder.

Para acceder a esta herramienta, nos situamos en la barra de menús de Quartus II, y se da clic en:

Tools-> SOPC Builder

Al iniciar SOPC Builder, debemos ingresar un nombre para el sistema que crearemos.

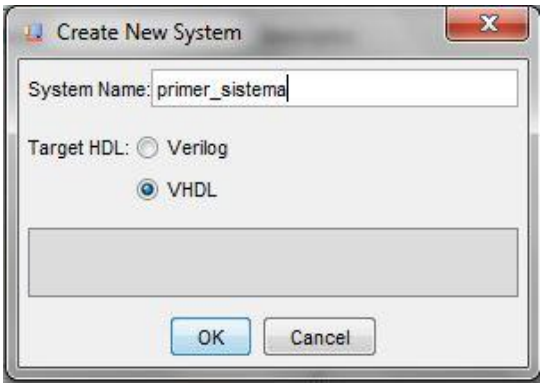


Figura 3-9 Ventana para la creación de un nuevo sistema en SOPC Builder.

Se abrirá entonces la pantalla principal de trabajo de SOPC Builder.

A la izquierda tenemos una sección con todas las librerías que el programa nos ofrece, con las que podemos agregar módulos (IP Cores) a nuestro sistema, como memorias, procesadores, puertos GPIO, otros.

3.4.1 COMPONENTES BÁSICOS DE UNA MÁQUINA SOPC

SOPC Builder permite al programador crear varios tipos de sistemas añadiendo varios módulos incluidos en sus librerías e inclusive módulos creados por terceros, sin embargo, los requisitos mínimos de una máquina SOPC son:

- **System ID:** El system ID es uno de los periféricos más importantes del sistema, ya que permite a las herramientas de desarrollo de

software para NIOS II saber si la aplicación desarrollada pertenece al hardware objetivo. Solo puede existir un periférico de este tipo por sistema y debe de llamarse sysid. Cualquier otro nombre provocará el no reconocimiento del periférico.

- **Procesador:** Hará las veces de cerebro para el sistema, ya que todo el software será interpretado por él y enviará las señales correspondiente a sus periféricos para completar las tareas asignadas.

- **Memoria On Chip:** Utilizaremos una memoria implementada dentro de la FPGA como espacio para almacenar el software del sistema así como los datos.

- **JTAG UART:** Éste módulo nos permite tener comunicación entre el computador y el chip. Si no incluimos éste módulo no podremos realizar algún cambio a la configuración del chip Cyclone II.

- **Puertos de Entrada y Salida (I/O):** Son necesarios porque de algún modo el sistema requiere obtener datos del exterior y luego de procesarlos mostrar o enviar como salida algún tipo de información útil para el usuario.

- **Timer:** Muchas aplicaciones requieren medir intervalos de tiempo para realizar acciones.

Cada vez que agreguemos un nuevo componente al sistema, SOPC Builder hará la interconexión entre cada uno de ellos y además, asignará direcciones de memoria a cada uno de ellos a menos que el desarrollador las defina manualmente.

Es importante mencionar que el SOPC Builder elige los nombres de los componentes automáticamente pero los nombres no son tan descriptivos para ser asociados o identificados con los elementos del proyecto pero se pueden cambiar dando clic derecho y seleccionando Rename.

3.4.2 CORES NECESARIOS PARA NUESTRO SISTEMA

Añadir el System ID

Peripherals-> Debug and Performance-> System ID peripheral (Renombrar un sysid)

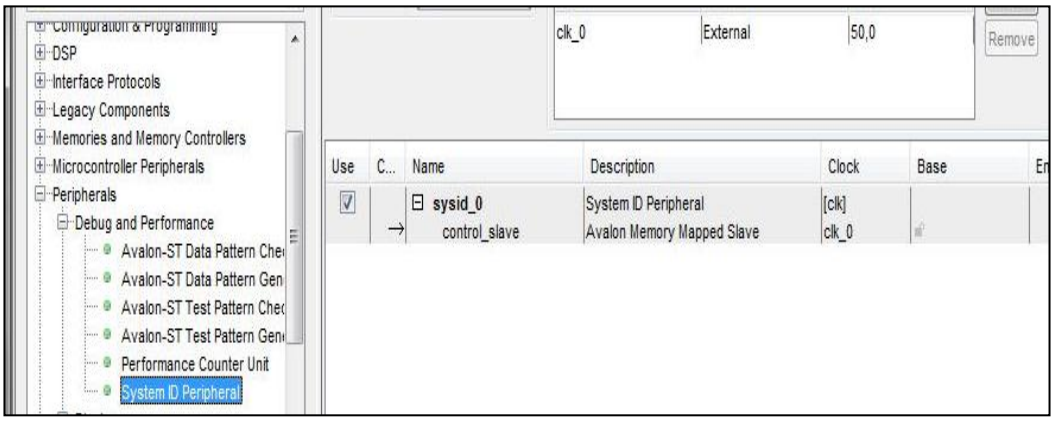


Figura 3-10 Añadir el System ID.

Agregar IP procesador

Processors-> Nios II Procesor (seleccionar Nios II/s). (Renombrar el CPU).

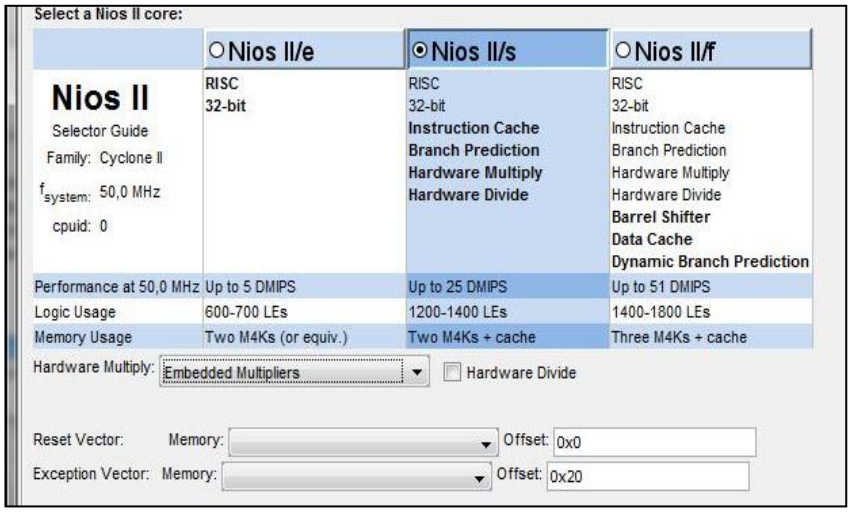


Figura 3-11 Agregar procesador NIOSII

Es necesario indicar al procesador cuáles son los módulos de memoria donde se ubicarán los vectores de Reset y Excepciones, pero como hasta ahora no hemos añadido un módulo de memoria, dejaremos esos valores sin configurar.

Agregar IP ON-CHIP-MEMORY

Memories and Memory Controllers -> On Chip -> On Chip Memory (ram o rom). Data width: 32, Total Memory Size: 4096Kbytes (Renombrar la memoria).

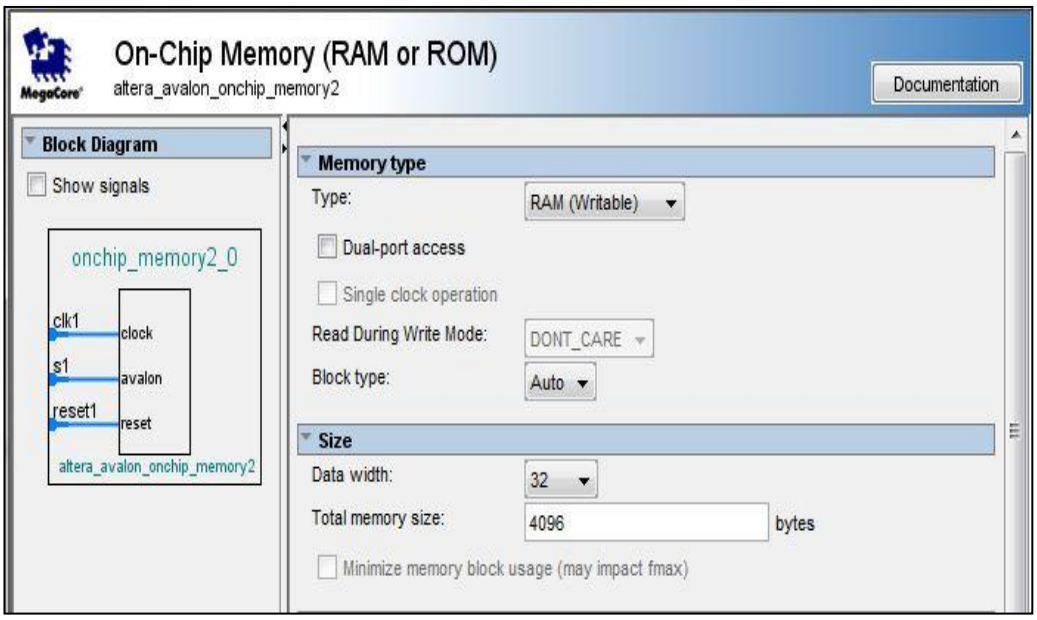


Figura 3-12 Asignación de memoria a nuestro sistema.

Ahora que hemos añadido una memoria a nuestro sistema, podemos regresar a la configuración del procesador dando doble clic sobre el módulo correspondiente y en la configuración de vector de Reset y de Excepción escogemos la memoria que acabamos de añadir.

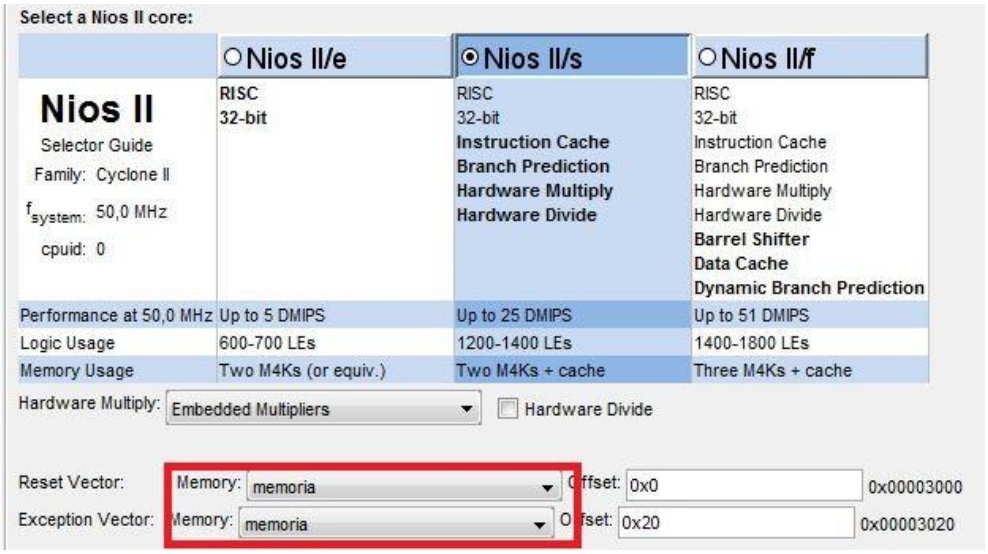


Figura 3-13 Inclusión de las memorias en el procesador.

Agregar IP Interval Timer

Peripherals-> Microcontroller Peripherals-> Interval Timer. Preset: full featured (Renombrar el timer)

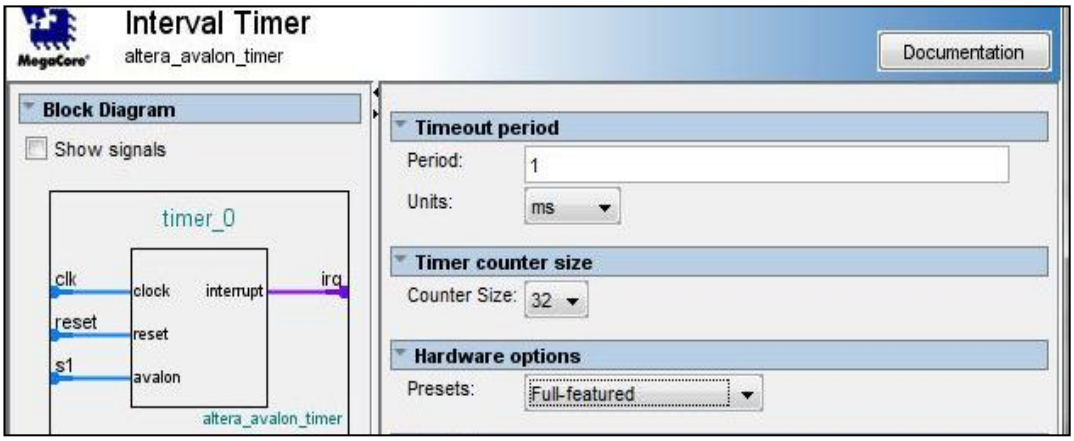


Figura 3-14 Agregar un interval timer.

Agregar IP Puerto paralelo de Entrada y Salida (PIO)

Peripherials-> Microcontroller Peripherials-> PIO. Elegir si lo necesitan como entrada, salida o bidireccional. Si necesitan varios pueden añadir más de 1 PIO.

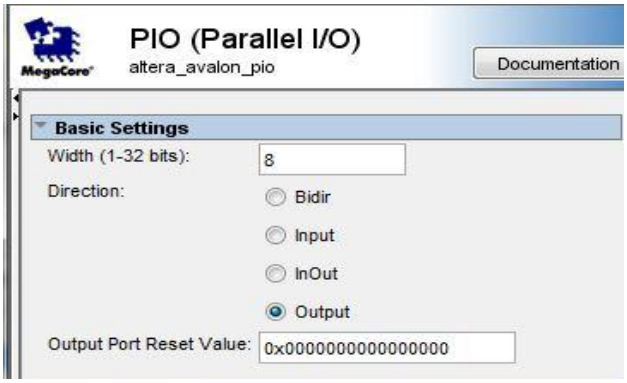


Figura 3-15 Agregar puertos de entrada/salida.

Agregar IP JTAG UART

Interface Protocols-> Serial-> JTAG UART (Renombrar el JTAG UART)

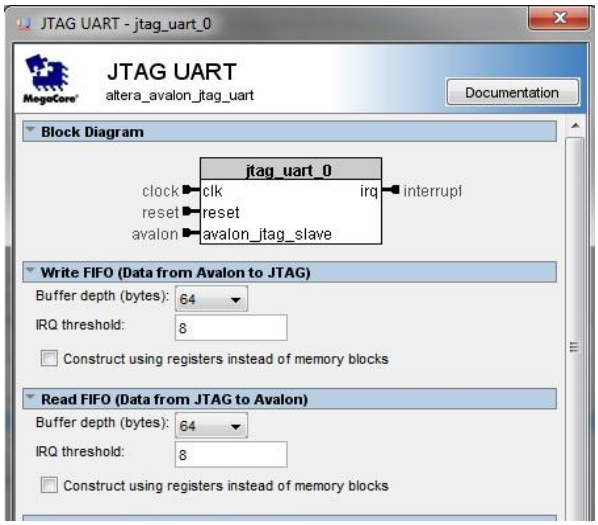


Figura 3-16 Asistente de configuración de la UART.

Nuestra lista de hardware debe lucir como lo muestra la siguiente figura:

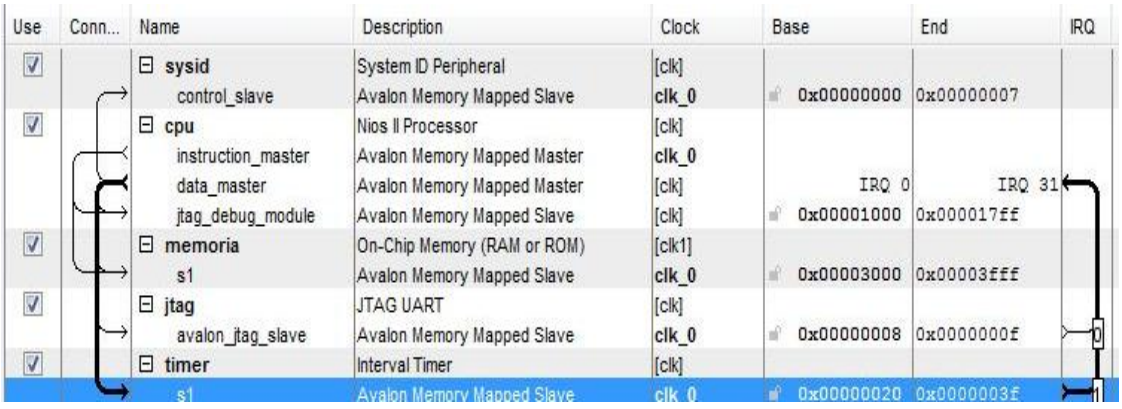


Figura 3-17 Lista de hardware del sistema.

El siguiente diagrama de bloques, muestra el hardware que hemos diseñado hasta ahora:

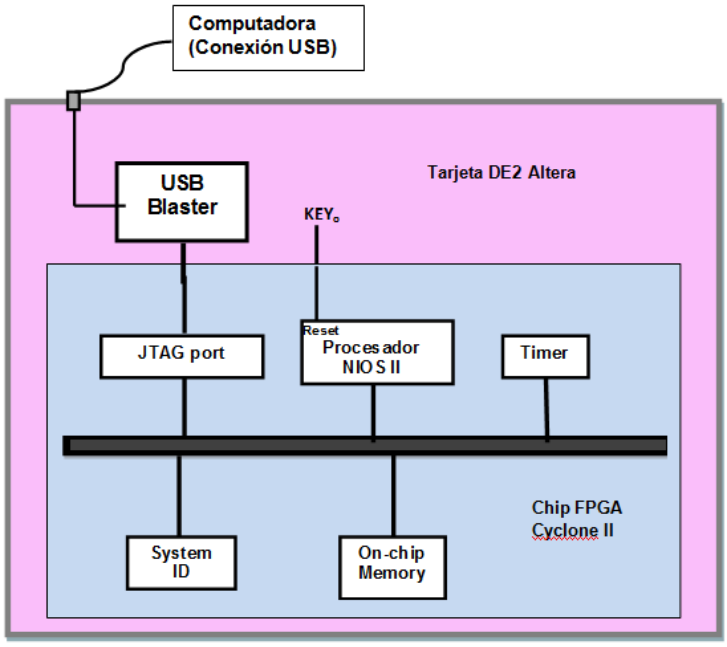


Figura 3-18 Diagrama de bloques de un sistema básico basado en NIOS II

Recodando que nuestro sistema es una calculadora y que anteriormente propusimos ingresar datos por medio de un teclado hexadecimal además de mostrar los datos en el display LCD 16x2 de la tarjeta DE2, notamos que aun debemos incluir los siguientes módulos o IP's:

- IP de LCD 16x2, para mostrar datos en el LCD 16x2 de la tarjeta DE2

- 1 IP GPIO, para usar el puerto de expansión JP2 donde se conectará el teclado hexadecimal.
- 1 IP GPIO, para utilizar los leds verdes como indicadores.
- 1 IP PIO (entrada), 1 IP PIO (salida), ambos usados para el funcionamiento del teclado hexadecimal.
- 2 IP PIO (como entrada) y 1 IP PIO (como salida), que agregaremos ahora a nuestro hardware llamándolos: sen, cos (entrada) y zeta (salida).

Agregar IP LCD 16x2

Peripherals-> Display-> LCD. (Renombrar LCD).

La pantalla correspondiente a agregar el IP LCD no muestra opciones de configuración por lo que resta solo presionar el botón FINISH para agregarlo.

Agregar IP GPIO para usar el módulo Green_leds

University Programa-> Generic-> Parallel Port. (Renombrar leds).

En I/O colocar que el dispositivo de salida serán los leds y abajo elegir el color verde.

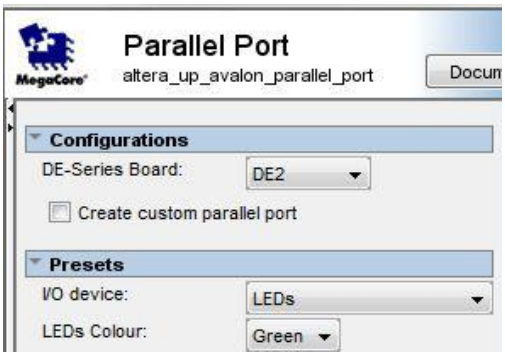


Figura 3-19 Añadir el módulo Green_leds

Agregar IP PIO para usar el Puerto de Expansión JP2

University Program-> Generic-> Parallel Port. (Renombrar a jp2).

En I/O colocar que el dispositivo será “Expansion Header” en JP2 .

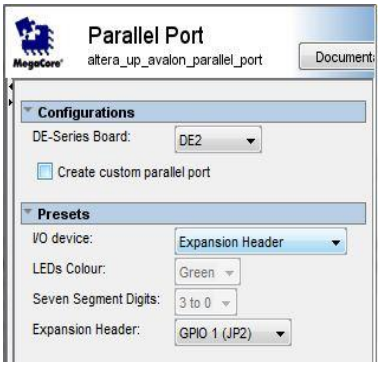


Figura 3-20 Añadir el Puerto de Expansión JP2

Agregar 2 IP PIO destinados a funcionamiento de teclado hexadecimal

Añadir un IP PIO de tipo input con 4 bits de ancho al que renombraremos **entrada**, y otro IP PIO de tipo output igualmente de 4 bits al que llamaremos **salida**.

Añadir 2 IP PIO de tipo input con 16 bits de ancho cada uno a los que llamaremos: **seno** y **coseno**, y otro IP PIO de tipo output con 17 bits de ancho al que llamaremos: **zeta**. Luego en SOPC Builder debemos ver una lista de hardware se muestra en la siguiente figura.

Use	C...	Name	Description	Clock
☑		⊞ sysid	System ID Peripheral	clk_0
☑		⊞ cpu	Nios II Processor	clk_0
☑		⊞ memoria	On-Chip Memory (RAM or ROM)	clk_0
☑		⊞ jtag	JTAG UART	clk_0
☑		⊞ timer	Interval Timer	clk_0
☑		⊞ lcd	Character LCD	clk_0
☑		⊞ leds	Parallel Port	clk_0
☑		⊞ jp2	Parallel Port	clk_0
☑		⊞ salida	PIO (Parallel I/O)	clk_0
☑		⊞ entrada	PIO (Parallel I/O)	clk_0
☑		⊞ seno	PIO (Parallel I/O)	clk_0
☑		⊞ coseno	PIO (Parallel I/O)	clk_0
☑		⊞ zeta	PIO (Parallel I/O)	clk_0

Figura 3-21 Lista de hardware en el SOPC Builder.

Luego de haber añadido los IP's necesarios para nuestro sistema, presionamos el botón GENERAR. Este proceso creará una descripción en lenguaje HDL de todos los IP que se agregaron al sistema SOPC Builder. El resultado será un concentrado de las características del sistema que podrá ser utilizado por QuartusII y por Nios II para continuar con el flujo de diseño. Ahora nuestro diagrama de bloques luciría así:

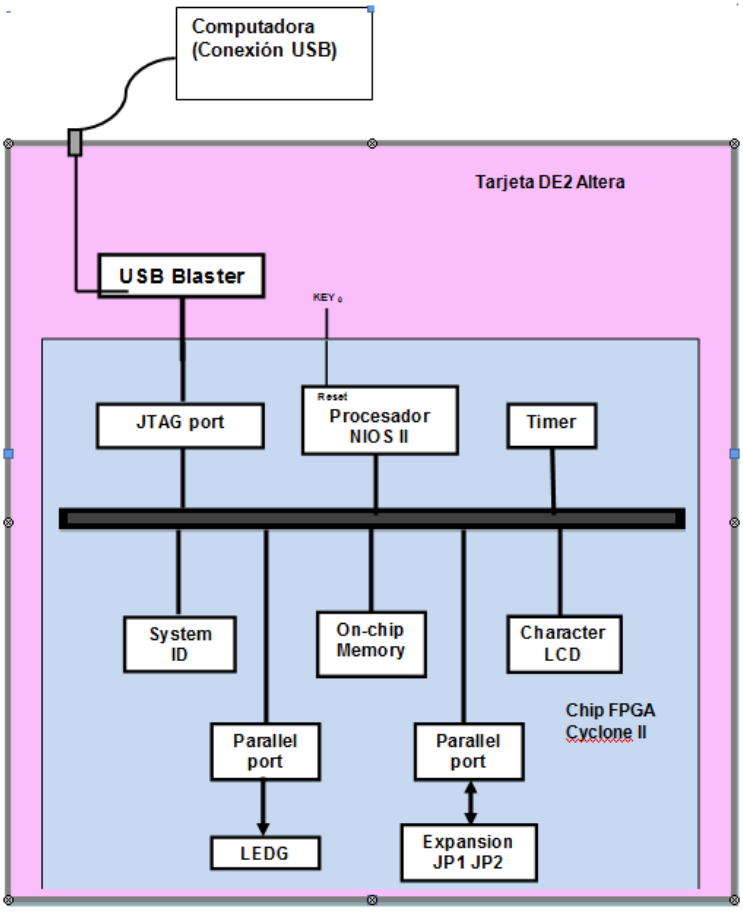


Figura 3-22 Diagrama de bloque de máquina calculadora basada en NIOS II

Si el proceso de generación no presenta errores, obtendremos un mensaje como el siguiente:

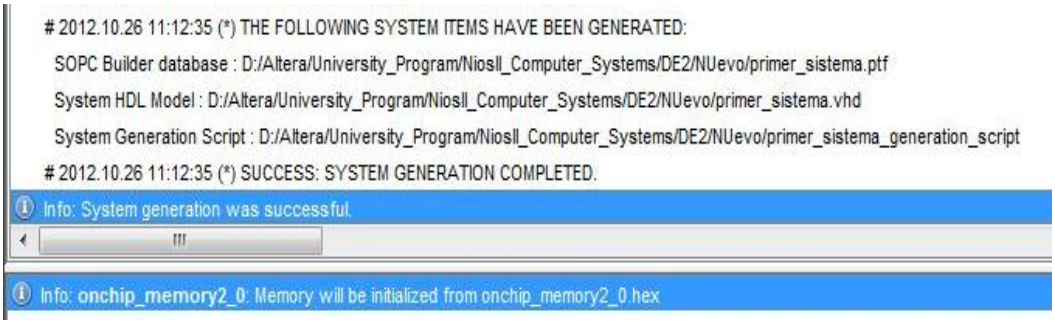


Figura 3-23 Mensaje de proceso de generación sin errores.

Luego de haber generado el hardware en SOPC Builder, cerramos dicha aplicación y regresamos a Quartus II donde continuaremos con el proceso normal en cualquier proyecto.

3.5 AÑADIR IP CORE BASADO EN EL ALGORITMO CORDIC A NUESTRO SISTEMA

Nuestro sistema debe realizar el cálculo de funciones senos, cosenos y tangentes, para lo cual vamos a utilizar un CORE basado en el algoritmo CORDIC. El core está escrito en lenguaje VHDL (la creación de CORES

utilizando lenguajes de descripción de hardware está fuera del alcance de este proyecto) y fue obtenido de la web www.opencores.org.

El core está formado por 3 archivos que deben ser agregados a nuestro proyecto en QuartusII.

Para añadir los archivos del core, seguiremos la siguiente ruta en QuartusII:

Project-> Add/Remove Files on a Project

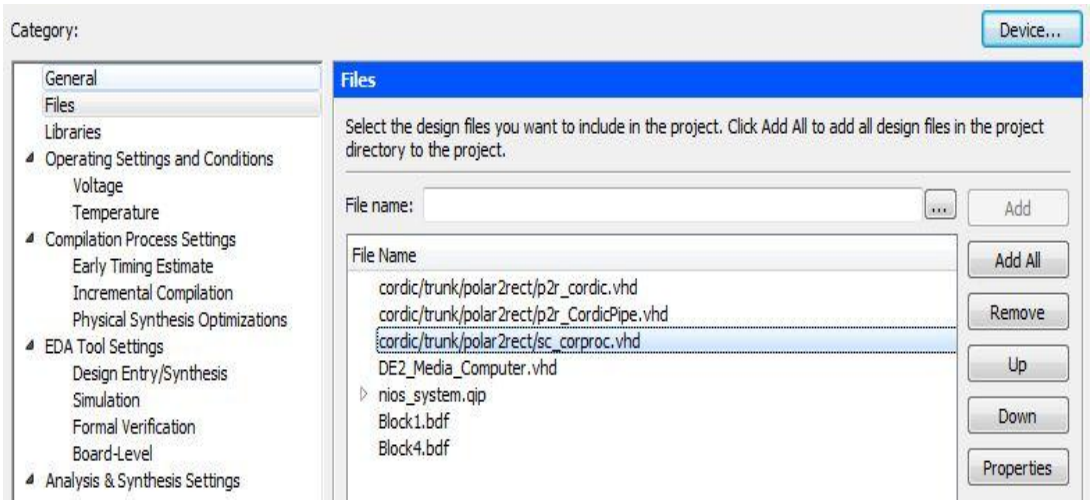


Figura 3-24 Agregar los archivos del core CORDIC.

Nuestro diagrama de bloques (simplificado) después de añadir un core a nuestro sistema se muestra en la Figura 3-25.

Las flechas entre el CORDIC core y los PIO seno, coseno y zeta deben ser interconectados como se muestra en el diagrama, siendo ZETA un

parámetro de entrada para el core y SENO, COSENO, valores de salida del core que luego deben ser leídos por el CPU del sistema.

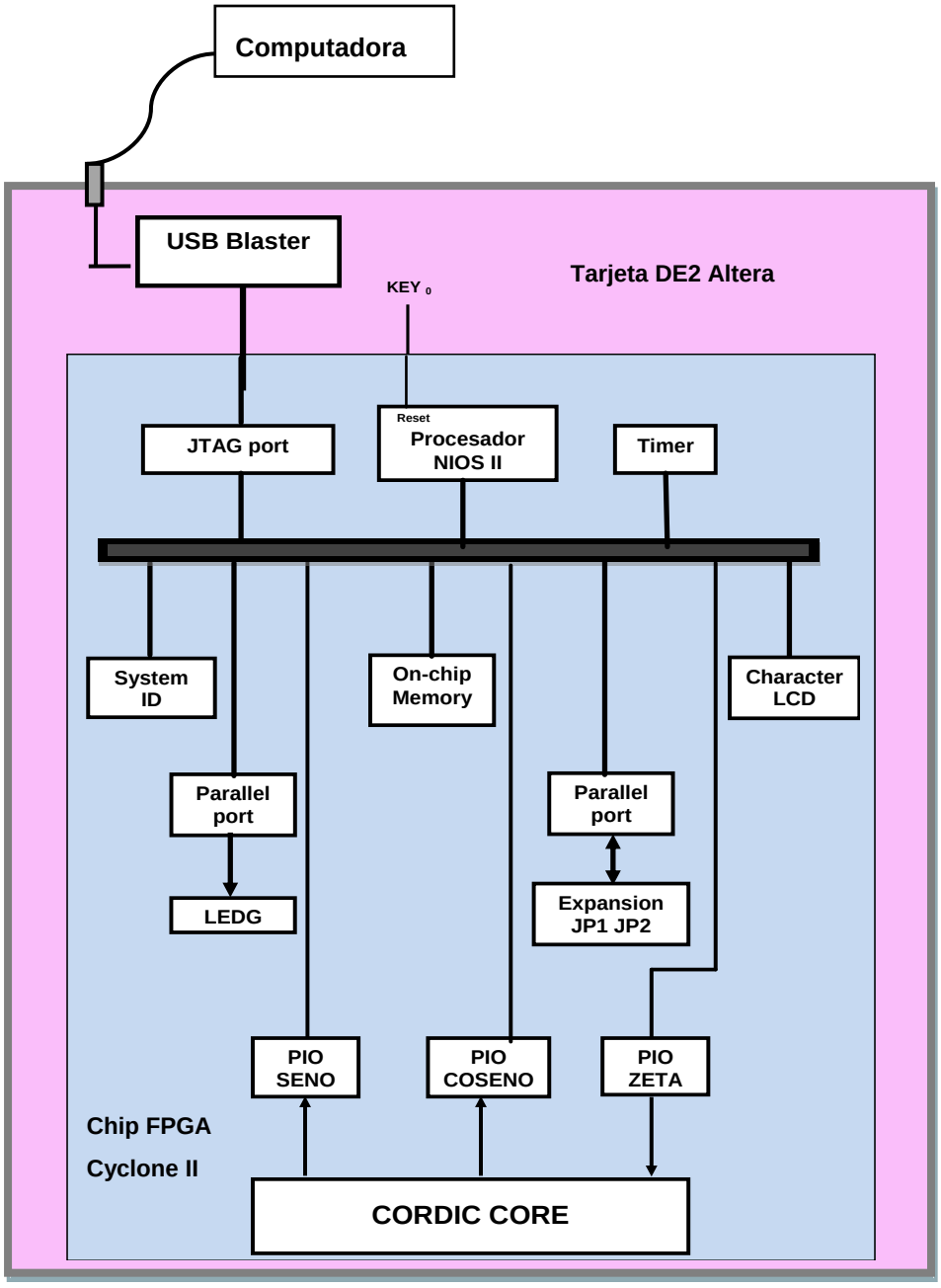


Figura 3-25 Diagrama de bloque del sistema con core CORDIC

Para realizar dicha conexión basta recordar el curso de SISTEMAS DIGITALES I y la programación en VHDL. Crearemos una nueva entidad con 2 componentes: nios_system (nuestro sistema) y sc_corproc (CORDIC core) y luego debemos unir ambos componentes con señales internas en los puertos correspondientes.

```
-------------------------------------------------------------
   component nios_system
      port (
   end component;


   |
   component sc_corproc is
      port(
   end component;

-------------------------------------------------------------
-------------------------------------------------------------
--            Internal Wires and Registers Declarations
-------------------------------------------------------------
-- Internal Wires
-- Used to concatenate some SDRAM control signals
signal           BA : STD_LOGIC_VECTOR(1 DOWNTO 0);
signal           DQM : STD_LOGIC_VECTOR(1 DOWNTO 0);

signal          wzeta: signed(16 downto 0);
signal          wsen: signed(15 downto 0);
signal          wcos: signed(15 downto 0);
```

Figura 3-26 Señales internas que serán usadas para interconectar los componentes nios_system (nuestro sistema) y el sc_corproc (CORDIC core).

Con esto, hemos añadido a nuestro diseño todos los componentes necesarios para lograr la funcionalidad de nuestro proyecto. Para la siguiente parte, regresaremos a Quartus II para seguir con el proceso de compilación.

3.6 COMPILACION EN QUARTUS II DEL SISTEMA DISEÑADO EN SOPC BUILDER

Regresando al entorno de trabajo de Quartus II, añadimos el esquemático de la máquina que acabamos de generar con SOPC Builder.

File-> New->Block diagram/Schematic File. Damos doble clic sobre la pantalla que aparece y nos debe aparecer la lista de proyectos disponibles, escogemos el nombre de nuestro proyecto y presionamos el botón OK.

Debe aparecer un bloque como el mostrado en la Figura 3-27, llamado primer_sistema, que representa a nuestra máquina diseñada en SOPC Builder y que posee varias terminales que corresponden a los módulos que añadimos.

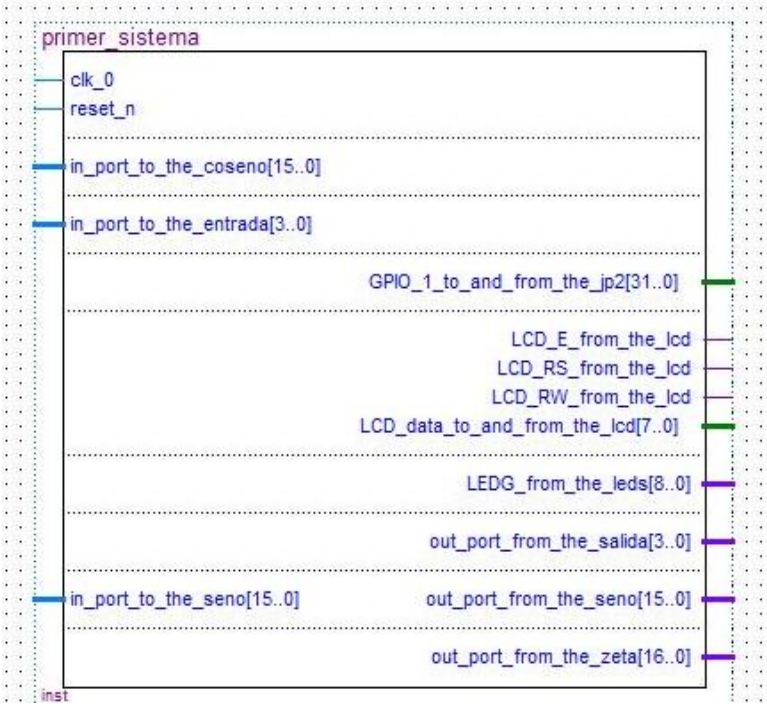


Figura 3-27 Diagrama que muestra el sistema diseñado en SOPC Builder

En dicho diagrama no aparecen los bloques adicionales de CORDIC que agregamos, debido a que ellos no fueron parte del procedo de GENERACION de archivo realizado en el programa SOPC Builder.

El siguiente paso es agregar pines físicos de la tarjeta a las terminales de nuestro diseño. Esto permite encaminar de manera correcta las señales generadas de manera interna del FPGA con sus correspondientes generadores /receptores fuera del dispositivo.

3.6.1 ASIGNACION DE PINES

- Processing-> Start-> Start Analisys and Elaboration

- Assigments-> Pin Planner. Es aquí donde debemos asignar el correspondiente pin a cada una de las terminales de nuestro diseño.

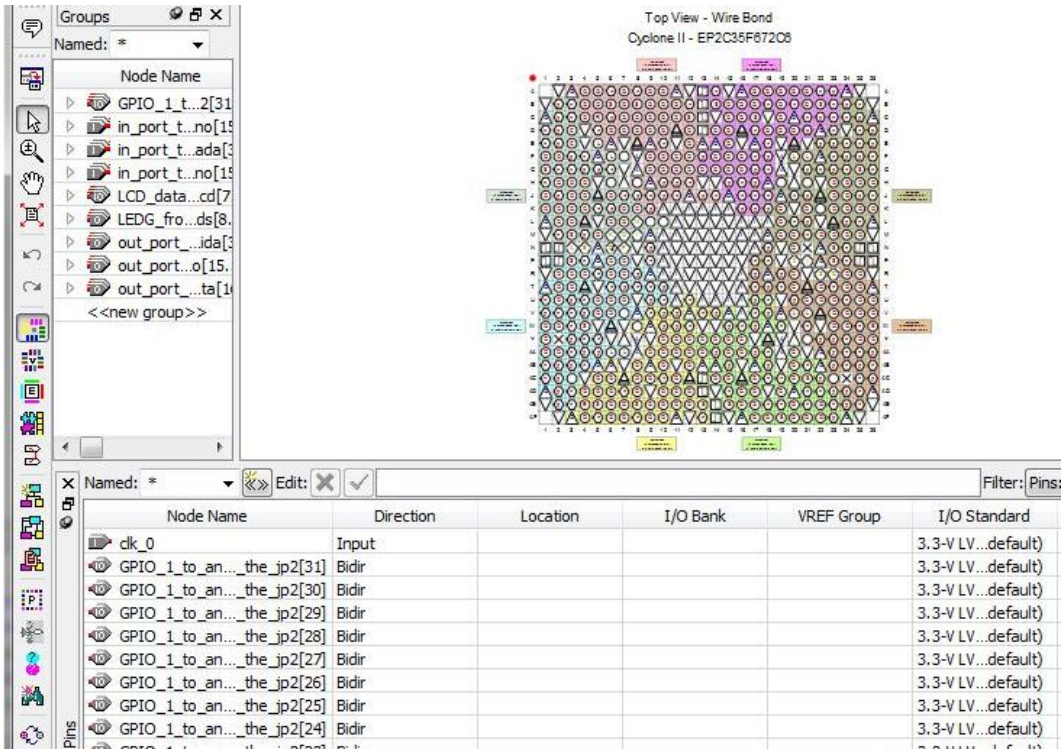


Figura 3-28 Ventana donde se realiza la asignación de pines.

La correspondencia de pines con componentes en la tarjeta Altera DE2 puede ser consultada en “DE2 Development and Education Board – User Manual”.

La asignación de pines realizada en nuestro proyecto puede ser revisada a continuación:

Nombre de la señal	**Dirección**	**Ubicación de PINES**	**Descripción**
ENTRADA[3]	Input	PIN_M21	Señales de salida del teclado, que son leídas por el procesador
ENTRADA[2]	Input	PIN_N20	
ENTRADA[1]	Input	PIN_M20	
ENTRADA[0]	Input	PIN_M19	
LEDS[3]	Output	PIN_V18	Señales indicadores a mostrarse en los leds verdes
LEDS[2]	Output	PIN_W19	
LEDS[1]	Output	PIN_AF22	
LEDS[0]	Output	PIN_AE22	
SALIDA[3]	Output	PIN_M23	Señales de entrada el teclado, enviadas desde el procesador
SALIDA[2]	Output	PIN_M22	
SALIDA[1]	Output	PIN_K26	
SALIDA[0]	Output	PIN_K25	
CLOCK_27	Input	PIN_D13	Reloj de 27Mhz
CLOCK_50	Input	PIN_N2	Reloj de 50Mhz

Tabla 3-1 Tabla con asignación de pines usados

Cuando se utilizan IP Cores listados en SOPC Builder, la asignación de pines correspondientes a cada IP se realiza de forma automática y solo es necesario realizar la asignación manual a señales específicas de nuestro propósito. Por ejemplo, cuando se añade el IP core correspondiente al display LCD, luego de generado el archivo, la asignación de pines correspondiente entre las interfaces del IP con las del módulo físico se establecen sin la intervención de usuario.

3.6.2 COMPILACIÓN

Para compilar nuestro proyecto de Quartus II hacemos clic en:

Processing-> Start Compilation. Esto inicia el proceso de compilación.

En caso de no haber errores en el proyecto, debemos el siguiente mensaje:

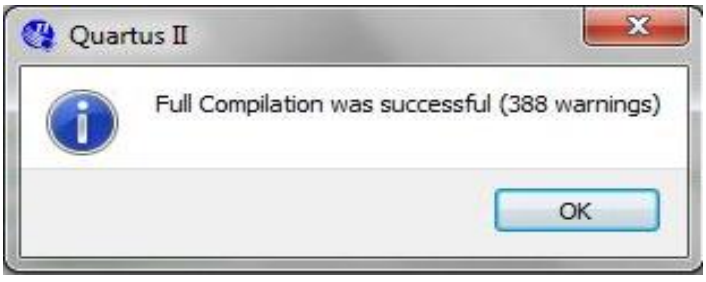


Figura 3-29 Ventana que aparece después de una compilación exitosa.

Una vez finalizado el proceso de compilación, podemos revisar un diagrama RTL que resultó de la compilación del programa. En él se muestran cada uno de los componentes que posee nuestra entidad.

Tools-> NetList Viewer-> RLT Viewer

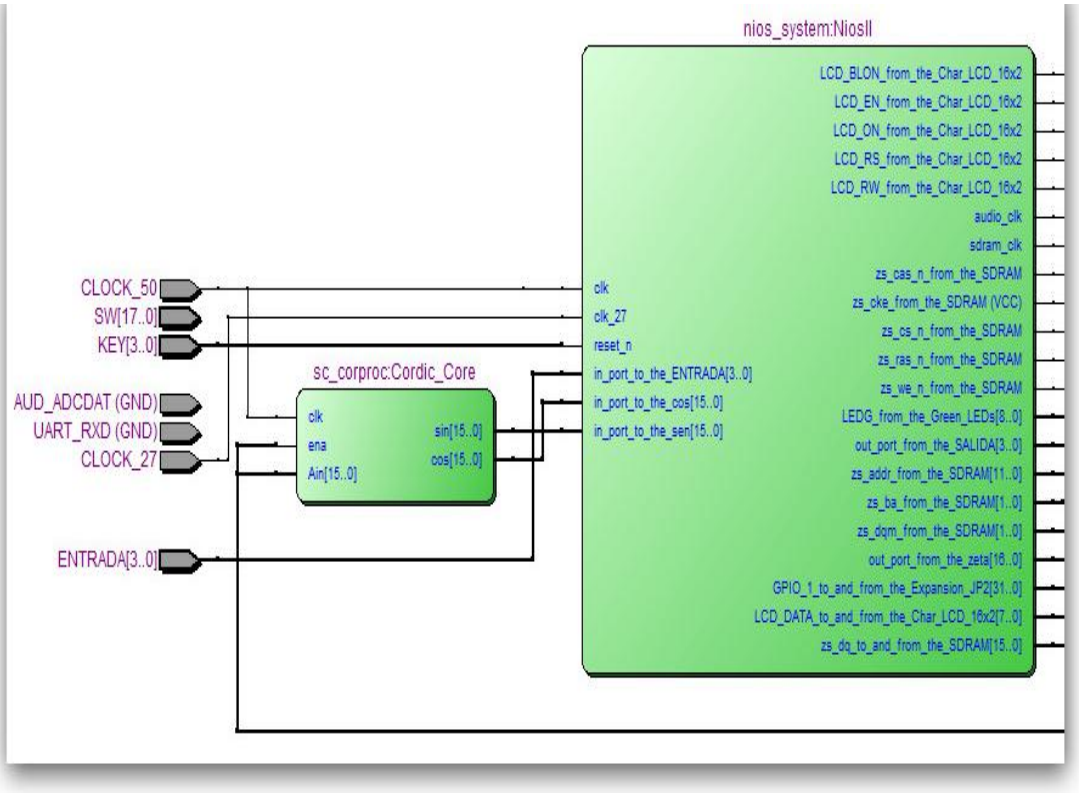


Figura 3-30 Diagrama obtenido en la herramienta RTL Viewer (Quartus II).

3.6.3 CARGA DEL DISEÑO AL FPGA

Después de que el diseño ha sido satisfactoriamente compilado, en Quartus II son generados los archivos necesarios para programar y configurar la FPGA.

- Tools-> Programmer. Se despliega el asistente para la programación de dispositivos.

Antes de iniciar la carga del diseño debemos revisar:

- Que se esté aplicando alimentación a la tarjeta DE2.
- La conexión USB entre el PC y el puerto USB Blaster de la tarjeta.
- Configuración de la tarjeta DE2 esté en modo JTAG.
- El archivo de diseño que va a ser volcado debe tener extensión .sof

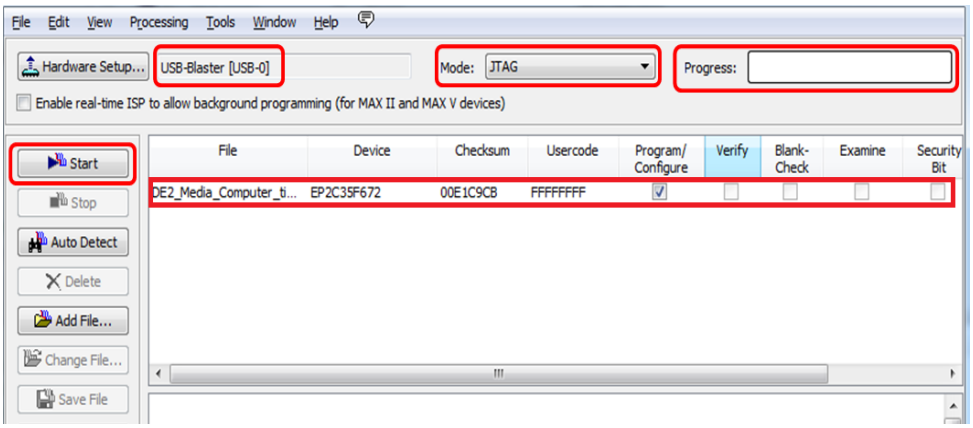


Figura 3-31 Ventana para cargar la computadora con NIOS II en el dispositivo.

Presionamos el botón START. Al terminar, la barra de progreso muestra 100%. En este momento la computadora basada en NIOS II está cargada en el dispositivo pero aún no es capaz de realizar ninguna tarea sin el software apropiado.

3.7 OBTENCIÓN E IDENTIFICACIÓN DE DATOS.

El teclado hexadecimal utilizado en este proyecto para la obtención de datos es una matriz de botoneras de dimensión 4x4, es decir, 16 botoneras en total. El teclado presenta 8 pines para su conexión, 4 pines son de ingreso de datos y los 4 pines restantes para salida de datos.

En los 4 pines de entrada del teclado se hace desplazar un voltaje bajo (0 voltios) a velocidad alta.

Los 4 pines de salida del teclado estarán conectados a voltaje alto (5 voltios) protegidos con resistencias. Cuando se presiona un botón del teclado se cierra el circuito de la botonera presionada de modo que se detecta un cambio en los pines de salida de 5 a 0 voltios, conociendo en que fila del teclado se encontraba el bajo voltaje y conociendo en que columna se detectó también el bajo voltaje es que se puede saber qué tecla se ha presionado. (Ver Figura 3-32).

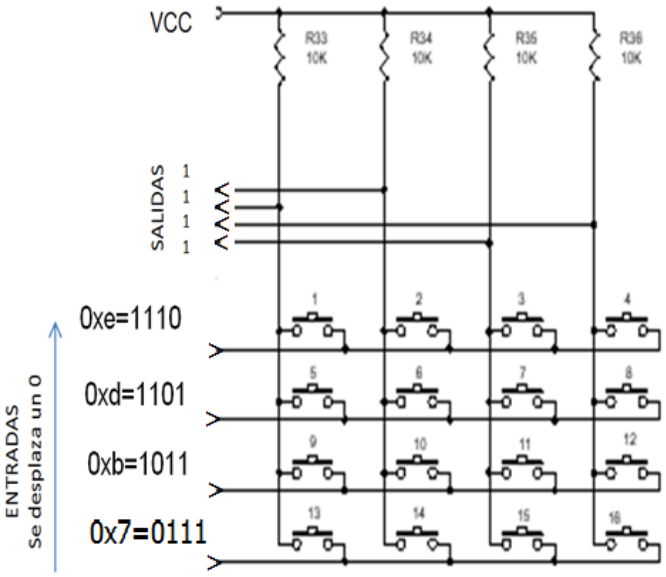


Figura 3-32 Diagrama del teclado cuando no se ha presionado alguna tecla

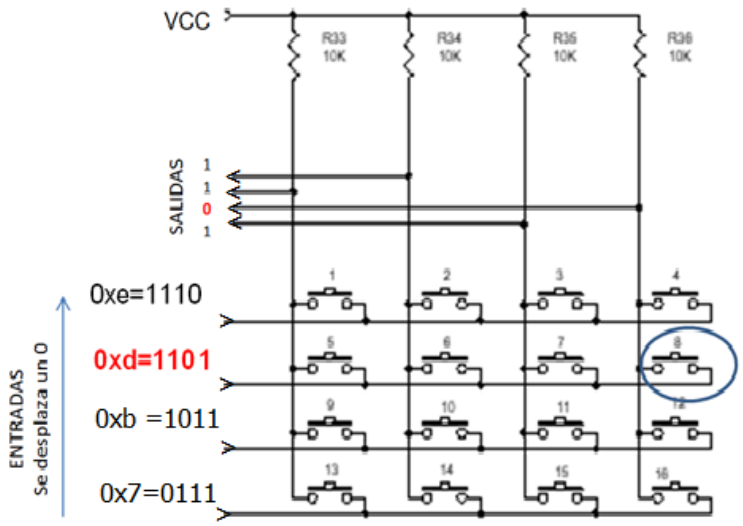


Figura 3-33 Diagrama que muestra el cambio en la salida al presionar una botonera.

Los valores de entrada al teclado, que hacen desplazar un 0 en una palabra de 4 bits, deben ser enviados desde el sistema basado en NIOS II hasta el teclado, para eso es necesario el IP PIO tipo salida de 4 bits, que creamos anteriormente llamado SALIDA. Los valores de voltaje que activan las filas de botoneras del teclado son enviados a través del PIO SALIDA.Los valores de salida del teclado, deben ser leídos y procesados por el sistema basado en NIOS II. El medio de transporte para esos datos es el IP PIO tipo entrada de 4 bits que incluimos anteriormente llamado ENTRADA.

Pero, ¿Cómo conectar los pines del teclado con dichos IP's?

El teclado está conectado al puerto de expansión JP2 de la tarjeta DE2 usando como medio un BUS de datos. A la computadora basada en NIOS2 se agregó el PIO SALIDA y el PIO ENTRADA (ver sección “Core necesarios para nuestro sistema”) que son los encargados del transporte de los datos transmitidos/recibidos por el teclado desde el procesador al teclado y viceversa. Cada bits de estos PIO son asignados a una ubicación de pines en la FPGA (ver sección” Asignación de Pines”).

3.8 CORES IMPLEMENTADOS

CORE BASADO EN ALGORITMO CORDIC

El core está construido en base a 3 bloques fundamentales. El pre-procesador, el post-procesador y el core CORDIC. Este último es construido usando un pipeline de bloques CordicPipe. Cada bloque CordicPipe representa un único paso en el proceso de iteración del algoritmo.

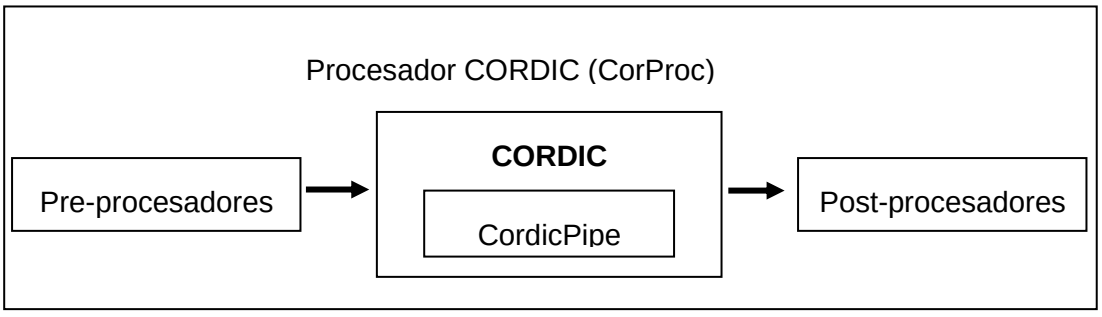


Figura 3-34 Diagrama de bloques fundamentales del core CORDIC.

Pre and Post procesadores

Debido a que la tabla de la arcotangente usada en el algoritmo CORDIC solo converge en el rango de -1rad a +1 rad, para usar el algoritmo CORDIC sobre el rango 2π, el dato de entrada necesita ser manipulado para caber en el rango de -1 a +1 rad. Esto es manejado por el pre-procesador. El post-procesador corrige esto y coloca los resultados del core CORDIC en el cuadrante correcto.

CORDIC Core

El core CORDIC es el corazón del núcleo del procesador ya que es el que realiza el algoritmo CORDIC. Todas las iteraciones son realizadas en paralelo usando la técnica de pipeline. Debido al uso de éste método el core puede realizar una transformación de CORDIC cada ciclo de reloj. Cada paso de la iteración del algoritmo es realizada por el core CordicPipe el cual contiene la tabla de valores de arcotangente (precalculada) para cada iteración.

Cálculo de Seno y Coseno

Los valores de Seno y Coseno para un valor de ángulo ingresado pueden ser calculados usando el siguiente escenario de CORDIC descrito en el capítulo 2:

$$[X_j, Y_j, Z_j] = [P(X_i \cos(Z_i) - Y_i \operatorname{sen}(Z_i)), P(Y_i \cos(Z_i) + X_i \operatorname{sen}(Z_i)), 0]$$

Y usando los siguientes valores como entrada para el core:

Xi=1/P=1/1.6467=0.60725
Yi=0
Zi=θ

El core calcula: $[X_j, Y_j, Z_j] = [\cos(\theta), \operatorname{sen}(\theta), 0]$

El valor de entrada Z toma valores de -180° a +180° donde:

0x8000= -180°
0xEFFF=+80°

Pero el core solo converge en el rango -90° a +90°.Las otras entradas y salidas están en el rango de -1 a +1 y la constante P representada en el capítulo 2 como K resulta en:

Xi=215.P=19898(dec)=4DBA(hex)

Ejemplo:

Para calcular el seno de 30°, el ángulo tiene que ser transformado como sigue:

360 deg = 2^{16}

1 deg= 2^{16}/360

30 deg = (2^{16}/360) **.** 30 = 5461(dec) = 1555(hex)

El valor de 5461(dec) o 1555(hex) es el valor de Z que ingresa al core CORDIC. El core lo procesa y arroja los siguientes resultados:

Sen= 16380(dec)=3FFC(hex)

Cos= 28381(dec)=6EDD(hex)

Las salidas representan valores en el rango -1 a +1. El resultado final puede ser obtenido como se muestra:

2^{15}=1.0 2^{15}=1.0

16380=1.0/2^{15}.16380=0.4999 28381=1.0/2^{15}.28381=0.8661

Mientras que el resultado esperado habría sido 0.5 y 0.866 respectivamente.

	0 deg	30 deg	45 deg	60 deg	90 deg
Sen	0x01CC	0x3FFC	0x5A82	0x6EDC	0x8000
Cos	0x8000	0x6EDD	0X5A83	0x4000	0x01CC
Sen	0.01403	0.49998	0.70709	0.86609	1.00000
Cos	1.00000	0.86612	0.70712	0.0000	0.01403

Tabla 3-2 Valores obtenidos en el cálculo de ángulos importantes haciendo uso del algoritmo CORDIC

Aunque el core es muy preciso, pequeños errores pueden ser introducidos al algoritmo (revisar la Tabla 3-2). Cuando se usa el core sobre el rango entero de salida, la diferencia entre +1 (0x7FFF) y -1 (0x8000) es sólo 1bit.

3.9 CREACIÓN DEL SOFTWARE EN NIOS II SBT.

Como primer paso debemos abrir la aplicación NIOS II SBT y crear un nuevo proyecto.

- File->New->Nios II Aplication and BSP from Template

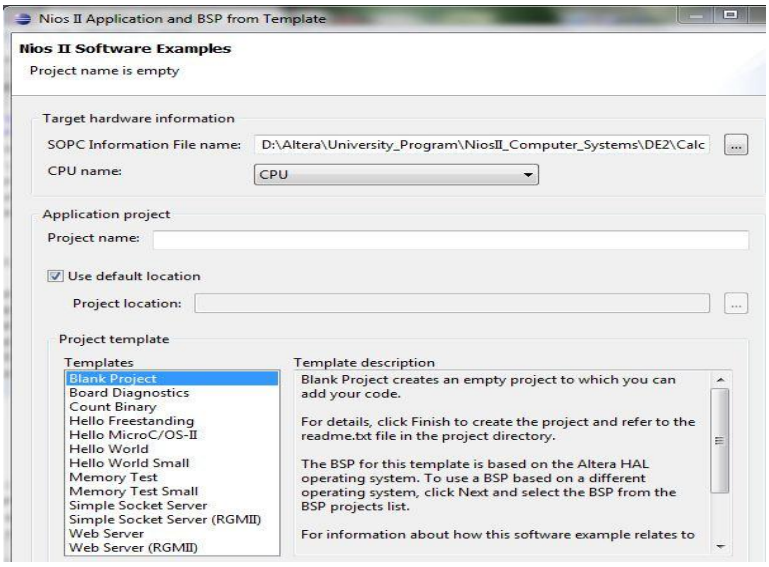


Figura 3-35 Ventana para crear un nuevo proyecto con NIOS II Aplication and SBT.

El proyecto de software se encuentra ligado al hardware diseñado en SOPC Builder como se muestra a continuación:

Agregamos el archivo de descripción de sistema generado por SOPC Builder (*.spocinfo). Seleccionamos el nombre del núcleo NIOS II agregado al sistema y luego colocamos un nombre para el proyecto. Como vamos a diseñar nuestro software desde cero, elegimos el template: Blank Template.

En la siguiente ventana, escogemos la opción: Create a new BSP Project based on the application Project template. Con esta opción el asistente se

encarga de dar información al proyecto de eclipse sobre que hardware se encuentra disponible para utilizar por el software del sistema.

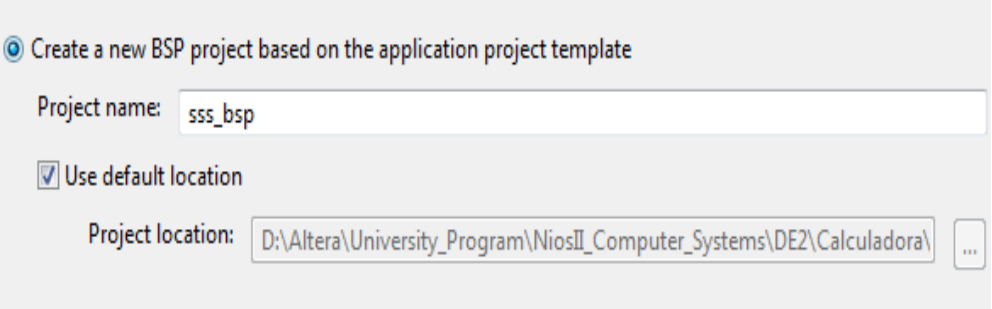


Figura 3-36 Ventana para crear un nuevo BSP.

Una vez creado un nuevo proyecto en blanco, vamos a crear un archivo nuevo (lenguaje C) al que llamaremos “principal.c” (Figura 3-37) y donde vamos a escribir el código de nuestra aplicación.

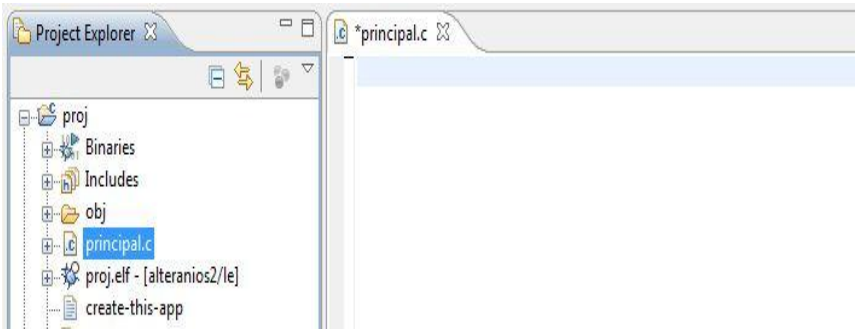


Figura 3-37 Ventana principal del proyecto.

Empezamos con las buenas prácticas de programación declarando varias librerías que serán útiles en el desarrollo de código y además la definición de varias constantes que en su mayoría representan direcciones de memoria de varios módulos de nuestro sistema creado en SOPC Builder, y a los cuales accederemos para leer o escribir datos.

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <io.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "altera_up_avalon_character_lcd.h"
#include "altera_up_avalon_character_lcd_regs.h"
#include <ctype.h>
#include <math.h>
#include <time.h>
```

Figura 3-38 Librerías usadas en el proyecto.

```
#define leg    0x10000010
#define gpio_in       0x00800000
#define gpio_out     0x00800010
#define  intToChar(entero) entero+0x30
#define zeta  0x00800040
#define sen   0x00800020
#define cos   0x00800030
#define z1    0x008000a0
#define z2    0x008000b0
```

Figura 3-39 Constantes definidas en el proyecto.

Luego declaramos los prototipos de cada función que hayamos implementado en el transcurso del programa. Explicaremos con detalle el funcionamiento de cada una de ellas.

```
int esdigito(char tec);
void lcd_clear_screen (alt_up_character_lcd_dev *lcd,int len,int delay1, int delay2);
char * convertIntChar(int num);
char leer_tecla();
int char2int(char c);
int string2int( char c[]);
char* dispTop(char fun,char signo,char numero[5],int digi);
char* dispBot(float res,char signo,int cuadrante);
int FuncionAin(int number);
int to_angulo_menor90(int angulo);
int Get_cuadrante(int angulo);
float resultado_funcion(int num_bin);
int binario_to_decimal(char resultado_binario[]);
char get_signo(int cuadro, char tecla);
int Get_cuadrante2(int angulo, char signo_in);
```

Figura 3-40 Lista de Funciones implementadas en el proyecto.

Esdigito: Función que recibe un carácter como parámetro y verifica si es un número o no.

Lcd_clear_screen: Función que borra los caracteres que se muestran en el LCD display

*** convertIntChar:** Función que convierte un número de 1 o varios dígitos en una cadena de caracteres.

Leer_tecla: Función que envía las señales que habilitan las filas del teclado, cuando una tecla es presionada, esta función analiza las señales de salida de la tecla y determina qué tecla se presionó.

Char2int: Función que convierte un número definido como carácter a entero.

String2int: Recibe una cadena de caracteres que represente un número y lo convierte a entero.

dispTop: Permite cambiar el mensaje de la línea de arriba en el LCD display

dispBot: Permite cambiar el mensaje de la línea de abajo en el LCD display

To_angulo_menor: Función que recibe un número entero de hasta 5 cifras que representa un valor de ángulo, y retorna un ángulo equivalente en el rango de 0 a 2 π.

Get_cuadrante. Esta función recibe un número entero de hasta 5 cifras que representa un valor de ángulo, y retorna el cuadrante donde se encuentra ubicado.

Binario_to_decimal: Recibe un número binario declarado como cadena de caracteres, y retorna el número entero correspondiente.

Get_signo: Esta función determina el signo que debe agregarse a la respuesta del cálculo de cualquier función seno, coseno o tangente, dependiendo del valor de ángulo ingresado y del cuadrante donde se encuentre ubicado.

INICIANDO EL PROGRAMA

Declaramos la función principal y algunas variables a usarse.

```
int main(){

	//punteros utilizados
	volatile long *p_zeta=zeta;
	volatile long *pleg=leg;
	volatile int *psen=sen;
	volatile int *pcos=cos;
	alt_up_character_lcd_dev * lcd;
	lcd =
alt_up_character_lcd_open_dev(CHAR_LCD_16X2_NAME);

	//enteros
	int len=15;
	int
valorsen=0,valorcos=0,dig=0,angulo_menor90,cuadrante;
	int ingresar=0, borrar=0, j=0, totalint=0,reducido=0;long
total=0;
	//caracteres y cadenas
	char tecla='q';
	char num[5]="qqqqq";
	char num2[5]="     ";
	char sig='a';
	char funcion='q';
```

El código del programa se desarrolló de acuerdo al diagrama mostrado en la Figura. 3-1 donde escogimos el funcionamiento para nuestra calculadora, de acuerdo a lo siguiente:

Estado 1

Se enciende la calculadora -> borra cualquier carácter en pantalla y mostrar el mensaje de bienvenida

```
alt_up_character_lcd_set_cursor_pos(lcd, 0, 0);
 alt_up_character_lcd_write(lcd, " <CALCULADORA> ", len);
 alt_up_character_lcd_set_cursor_pos (lcd, 0, 1);
 alt_up_character_lcd_write(lcd, "  Sen&Cos&Tan  ", len);
 usleep(2000000);
 lcd_clear_screen(lcd,16,5,5);
```

Estado 2

Luego de presionar ENTER en la pantalla de bienvenida, se cambia el mensaje de pantalla, se activan las señales que habilitan las filas del teclado y esperamos a que el usuario ingrese la función que se desea calcular (el sistema identifica la tecla presionada). Si la tecla presionada corresponde a una función, mostramos en pantalla SEN, COS o TAG respectivamente, caso contrario seguiremos esperando que ingrese la función.

Solo se puede pasar de este estado cuando el usuario haya presionado una tecla correspondiente a una función y seguido de presionar la tecla ENTER, con lo que se encenderá un led verde.

```
while(1){
   *pleg=0;
      while(1){
         if(x==0)
         {

                  alt_up_character_lcd_set_cursor_pos(lcd, 0, 1);
                  alt_up_character_lcd_write(lcd,"Valor:        ", len);
                  alt_up_character_lcd_set_cursor_pos(lcd, 0, 0);
                  alt_up_character_lcd_write(lcd,"Funcion:      ", len);
                  alt_up_character_lcd_set_cursor_pos(lcd, 0, 0);
                  alt_up_character_lcd_write(lcd,"Funcion: ", 9);
         }
         tecla=leer_tecla();
         if(tecla=='s' || tecla=='c' || tecla=='t' || tecla=='e'){
                  if(tecla=='s')
                  {
                           //************************ FUNCION        SENO *********************
                           alt_up_character_lcd_set_cursor_pos(lcd, 0, 0);
                           alt_up_character_lcd_write(lcd,"Funcion: Sen() ", len);
                           funcion=tecla;
                  x=1;
                  }else if(tecla=='c')
                  {
                           //************************ FUNCION COSENO************************
                           alt_up_character_lcd_set_cursor_pos(lcd, 0, 0);
                           alt_up_character_lcd_write(lcd,"Funcion: Cos() ", len);
                           funcion=tecla;
                  x=1;
                  }else if(tecla=='t')
                  {
                           //************************ FUNCION TANGENTE ********************
                           alt_up_character_lcd_set_cursor_pos(lcd, 0, 0);
                           alt_up_character_lcd_write(lcd,"Funcion: Tan() ", len);
                           funcion=tecla;
                  x=1;
               }
                  else if(tecla=='e'){
                           if(x==1)
                                    x=2;
                  }
         }
         if(x==2)
                  break;
      }//-> Hasta aquí, última revisión el 18 mayo 2013
      // FUNCION INGRESADA
      *pleg=1;// primera señal
```

Estado 3

Ahora se espera que se ingrese un valor de ángulo (número de hasta 5 cifras, entero, negativo o positivo). Cada vez que se ingresa un nuevo dígito se actualiza en pantalla el número total. Es posible utilizar la tecla DEL para borrar el último dígito del número en pantalla. Una vez que el número haya sido ingresado por el usuario, se presiona la tecla ENTER para mandar a calcular, y un nuevo led es encendido.

```
*pleg=1;// primera señal
ingresar=1;
borrar=0;
alt_up_character_lcd_set_cursor_pos(lcd, 0, 1);
alt_up_character_lcd_write(lcd,"Valor:         ", len);
alt_up_character_lcd_set_cursor_pos(lcd, 0, 1);
alt_up_character_lcd_write(lcd,"Valor:   ",9);

while(1)
{
    tecla=leer_tecla();
    if(esdigito(tecla) && dig<5){       // ingresar hasta máximo 5 dígitos
            num[dig]=tecla;
                        dig=dig+1;
                        for (j=0;j<5;j++){
                                if(num[j]!='q')
                                        num2[j]=num[j];
                        }
                        //mostrar numero en pantalla
                        alt_up_character_lcd_set_cursor_pos(lcd, 0, 1);
                        alt_up_character_lcd_write(lcd,"Valor:         ", len);
                        if(sig=='a'){
                                alt_up_character_lcd_set_cursor_pos(lcd, 9, 1);
                                alt_up_character_lcd_write(lcd,"+",1);
                        }
                        else if(sig=='r'){
                                alt_up_character_lcd_set_cursor_pos(lcd, 9, 1);
                                alt_up_character_lcd_write(lcd,"-",1);
                        }
                        alt_up_character_lcd_set_cursor_pos(lcd, 10, 1);
                        alt_up_character_lcd_write(lcd,num2,dig);
    }else if(tecla=='m'){
```

```
        }else if(tecla=='m'){
                // cambiar signo
                if(sig=='a')
                                        sig='r';
                                else if(sig=='r')
                                        sig='a';
                                //mostrar numero en pantalla
                                alt_up_character_lcd_set_cursor_pos(lcd, 0, 1);
                                alt_up_character_lcd_write(lcd,"Valor:         ", len);
                                if(sig=='a'){
                                        alt_up_character_lcd_set_cursor_pos(lcd, 9, 1);
                                        alt_up_character_lcd_write(lcd,"+",1);
                                }
                                else if(sig=='r'){
                                        alt_up_character_lcd_set_cursor_pos(lcd, 9, 1);
                                        alt_up_character_lcd_write(lcd,"-",1);
                                }
                                alt_up_character_lcd_set_cursor_pos(lcd, 10, 1);
                                alt_up_character_lcd_write(lcd,num2,dig);
                                //fin mostrar numero en pantalla
        }else if(tecla=='l'){
                // borrar ultimo digito
                if(dig>0){
                num[dig-1]="q";
                num2[dig-1]=" ";
                dig=dig-1;

                if(sig=='a'){
                        alt_up_character_lcd_set_cursor_pos(lcd, 9, 1);
                        alt_up_character_lcd_write(lcd,"+",1);
                }
                else if(sig=='r'){
                        alt_up_character_lcd_set_cursor_pos(lcd, 9, 1);
                        alt_up_character_lcd_write(lcd,"-",1);
                }
                alt_up_character_lcd_set_cursor_pos(lcd, 10, 1);
                alt_up_character_lcd_write(lcd,"     ",5);
                alt_up_character_lcd_set_cursor_pos(lcd, 10, 1);
                alt_up_character_lcd_write(lcd,num2,dig);

                }
        }else if(tecla=='e'){
                total=string2int(num);
                totalint=total;
                if(dig>0)
                        x=10;
        }
        if(x==10)
                break;
    }
    //   SEGUNDA SEÑAL //////////////////////
*pleg=3;
```

Estado 4

El número o valor de ángulo ingresado es verificado por las funciones correspondientes.

```
////// SEGUNDA SEÑAL
      *pleg=3;
      //validar angulo
      cuadrante=Get_cuadrante2(total,sig);
      printf("Ingreso: %c %d\n",sig,total);
      signo=get_signo(cuadrante,funcion);
      total=to_angulo_menor90(total);
      reducido=total;
      printf("angulo reducido: %d, cuadrante: %d, signo: %c \n ",total,cuadrante,signo);
```

Estado 5

Se prepara el valor de ángulo para ser enviado al CORDIC Core, y como fue explicado anteriormente, deben realizarse las siguientes operaciones para obtener el valor de Z, que es el dato de entrada para el cálculo de la función.

Primero, el ángulo tiene que ser calculado como sigue:

360 deg = 2^{16}

1 deg= $2^{16}/360$

30 deg = ($2^{16}/360$) **.** 30 = 5461(dec) = 1555(hex)

El valor de 5461(dec) o 1555(hex) es el valor de Z que ingresa al core CORDIC

```
//1. Calcular Z en decimal
        total=(total*(pow(2,16)))/360;

//3. Enviar enable
        *p_zeta=65536; //activado el bit(17)

//4. Enviar la función
        total=total+65536; // se le añade el bit de activación del core
        *p_zeta=total;
```

La pantalla del LCD se limpia para mostrar el mensaje respectivo (sen, cos o tag) y la respuesta calculada. Además se enciende un nuevo led como indicador de que la respuesta está siendo mostrada.

```
valorsen=*psen;
        valorcos=*pcos;
        float senf,cosf;
        senf=resultado_funcion(valorsen);
        cosf=resultado_funcion(valorcos);

        lcd_clear_screen(lcd,16,5,5);

if(funcion=='s'){
                *linea1=dispTop('s',sig,num2,dig);
        }else if(funcion=='c'){
                *linea1=dispTop('c',sig,num2,dig);
        }else if(funcion=='t'){
                *linea1=dispTop('t',sig,num2,dig);
        }
        alt_up_character_lcd_set_cursor_pos(lcd, 0, 0);
        alt_up_character_lcd_write(lcd,linea1, len);


if(funcion=='s'){
                *linea2=dispBot(senf,signo,cuadrante);
                alt_up_character_lcd_set_cursor_pos (lcd, 0, 1);
                alt_up_character_lcd_write(lcd,linea2, len);
        }else if(funcion=='c'){
                *linea2=dispBot(cosf,signo,cuadrante);
                alt_up_character_lcd_set_cursor_pos (lcd, 0, 1);
                alt_up_character_lcd_write(lcd,linea2, len);
```

Para casos especiales en los cálculos (por ejemplo: tangente (90)) se incluyen las siguientes líneas que muestran un mensaje de error inmediatamente evitando que el sistema procese en dichos escenarios.

```
}else if(funcion=='t'){
                float tagf=senf/cosf;

                while(totalint>=360){
                        totalint=totalint-360;
                }
                if(totalint==90 || totalint==270){
                        alt_up_character_lcd_set_cursor_pos (lcd, 0, 1);
                        alt_up_character_lcd_write(lcd,"Math Error     ", len);
                }
                else if(totalint==0 || totalint==180){
                        alt_up_character_lcd_set_cursor_pos (lcd, 0, 1);
                        alt_up_character_lcd_write(lcd,"0              ", len);
                }

                else{
                        *linea2=dispBot(tagf,signo,cuadrante);
                        alt_up_character_lcd_set_cursor_pos (lcd, 0, 1);
                        alt_up_character_lcd_write(lcd,linea2, len);
                }
        }
        *pleg=7;.
```

Estado 6

El sistema permanecerá mostrando el resultado calculado hasta que se presione la tecla ENTER, acto seguido se limpian los caracteres del LCD display, se apagan los leds verdes indicadores y programa vuelve al inicio del lazo while, empezando una nueva iteración permitiendo el cálculo de una nueva función con un nuevo valor.

```
 do{
          tecla=leer_tecla();

 }while(tecla!='e');
 alt_up_character_lcd_set_cursor_pos (lcd, 0, 0);
alt_up_character_lcd_write(lcd,"               ", len);
alt_up_character_lcd_set_cursor_pos (lcd, 0, 1);
alt_up_character_lcd_write(lcd,"               ", len);
usleep(500000);
*pleg=8;
alt_up_character_lcd_set_cursor_pos(lcd, 0, 0);
 alt_up_character_lcd_write(lcd, "  CALCULADORA  ", 15);
  alt_up_character_lcd_set_cursor_pos (lcd, 0, 1);
  alt_up_character_lcd_write(lcd, "  Sen-Cos-Tan  ", len);
 usleep(1500000);
 x=0;
 dig=0;
 tecla='q';

 for(j=0;j<5;j++){
      num[j]='q';
      num2[j]=' ';
  }
 j=0;
 sig='a';
 signo='+';
 funcion='q';
 *linea1="";
 *linea2="";
 cuadrante=1;
 printf("vuelta!\n");
    return 0;
 } // Fin de la función principal
```

El código completo de proyecto, incluyendo las funciones implementadas, se adjunta a este documento.

COMPILACION DEL CODIGO FUENTE

La compilación de código fuente anteriormente diseñado, permite la verificación de la validez del código (léxico, sintáctico y semántico), además de generar un archivo en código máquina el cual puede ser interpretado por el microprocesador NIOS II.

Project-> Build Project

Esta acción nos permite verificar el funcionamiento de nuestro sistema ejecutándose directamente en el hardware. Para cargar el programa en la FPGA es necesario que primero se haya cargado el hardware diseñado en SOPC Builder.

Figura 3-41 Compilar el proyecto.

3.10 INTERFAZ DE USUARIO

La interfaz guiará al usuario en el flujo de trabajo del programa una vez cargado en el hardware. Esta aplicación se basa el diagrama de la Figura 3-1 que explica el funcionamiento establecido para nuestra calculadora.

Cuando se enciende la calculadora, se presentará en el display el mensaje:

****CALCULADORA****

Luego de 2 segundos, el mensaje del display será:

FUNCION:

Donde el usuario primero debe presionar las teclas de SEN, COS o TAG, para indicar al programa la función que quiere calcular, no es posible realizar el cálculo de varias funciones a la vez de acuerdo al diseño de la interfaz, y debe primero ingresarse la función a calcular antes que el valor del ángulo de entrada.

FUNCION: SENO

Cada vez que el usuario presiona las teclas de funciones, cambia la palabra en el display (SENO, COSENO o TANGENTE). Una vez escogida la función,

se debe presionar el botón ENTER. En este paso, un LED VERDE se enciende, indicando que la función ya ha sido ingresada.

A continuación, se debe ingresar el valor de ángulo para realizar el cálculo:

SEN (-30) =

Pueden ser ingresados números enteros de hasta 5 cifras, y además el usuario puede presionar la tecla DEL para borrar el último dígito del número ingresado, además se puede cambiar el signo del valor ingresado presionando la tecla +/-.

Una vez ingresado el valor del ángulo, el usuario debe presionar la tecla ENTER para realizar el cálculo. Luego, se encenderá otro LED VERDE indicando que el valor de ángulo ya fue ingresado y se va a realizar el cálculo.

El programa analiza el valor del ángulo ingresado, si es un ángulo mayor a 2π el programa lo reduce a su ángulo correspondiente en el intervalo de 0 a 2π, luego analiza en que cuadrante se encuentra el ángulo y realiza el cálculo respectivo retornando la respuesta con el signo correspondiente.

La pantalla permanecerá mostrando la respuesta hasta que el usuario presione la tecla ENTER, y el programa volverá al mensaje de inicio. Además los LEDS VERDES se apagarán y el proceso se repite.

CAPÍTULO 4

4 PRUEBAS

Esta sección contiene los datos y resultados de varias pruebas realizadas sobre nuestro proyecto con el diseño y hardware que se detalló en los capítulos anteriores.

Las pruebas realizadas tienen como objetivo la obtención del tiempo de convergencia del core CORDIC implementado (es decir, el tiempo que toma el core en procesar los datos de entrada y retornar valores en su salida), para luego compararlo contra el **tiempo que tomarían en procesar otras implementaciones del algoritmo CORDIC.**

4.1 PRELIMINARES

Nuestro diseño de sistema detallado en el capítulo 3 nos muestra dos bloques principales: el bloque que representa al core CORDIC y el bloque correspondiente al Sistema NIOS, ambos funcionan con la misma velocidad de reloj (Clock_50), por lo que el análisis del CORDIC Core y el tiempo que

toma en procesar un dato de entrada podemos hacerlo por separado creando un nuevo proyecto en Quartus II incluyendo solamente los archivos del core.

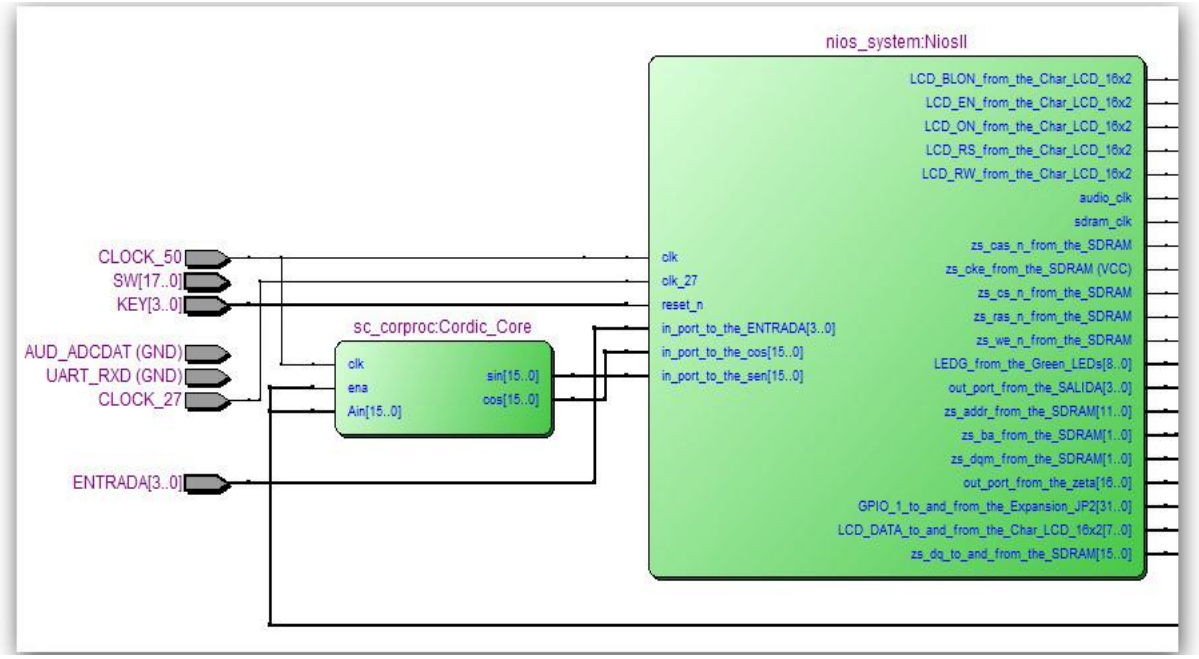


Figura 4-1 Diagrama de Bloque de nuestro sistema

4.2 ANALISIS DEL CORE CORDIC Y MEDICIÓN DE SU TIEMPO DE CONVERGENCIA

Creamos el nuevo proyecto en Quartus II, añadimos a dicho proyecto los archivos del core y creamos un nuevo archivo esquemático al cual añadimos el bloque correspondiente al core.

Además vamos a asignar pines de la tarjeta de DE2 a las señales del core de tal modo que:

- Los valores asignados en los switches del 15 al 0 serán los valores de entrada Ain[15-0] del core (un dato binario de 16 bits)
- El valor asignado en el switch 17 será el valor de la señal habilitadora (ena) del core (un dato binario de 1 bit).
- Los valores de salida de la señal COS[15-0] se reflejarán en los leds rojos del 15 al 0 (un dato binario de 16 bits).

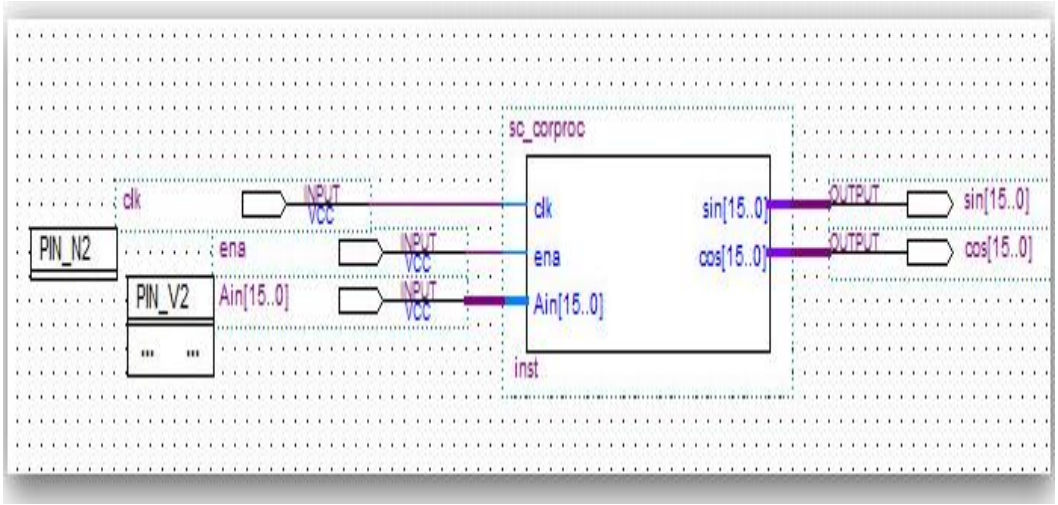


Figura 4-2 Diagrama esquemático del core

El objetivo de éste método es poder medir el número de flancos de reloj necesarios para que el CORDIC core procese el dato de entrada una vez

que se envía la señal habilitadora (ena o enable). Se sabrá que el core ha finalizado de procesar el dato cuando los leds rojos cambien, es decir, cuando se presente un nuevo número binario a la salida del core. Es fácil de notar que no se precisa trabajar con una velocidad específica de reloj, ya que una vez que obtengamos la cantidad de flancos de reloj utilizados por el core, podemos obtener el "tiempo" que tomó en procesar de acuerdo a la siguiente ecuación:

Tiempo_proceso=(Numero_flancos)*(Periodo_reloj)

Para contar el número de flancos que toma el core en procesar un dato podemos:

- Realizar una simulación de nuestro diseño esquemático y analizando un diagrama de tiempo (obtenido en la simulación), obtenemos el número de flancos de reloj usados para el proceso del CORDIC Core.

- Si la velocidad de reloj de la tarjeta DE2 fuera más lenta, podríamos utilizar un contador para obtener los flancos de reloj necesarios hasta que el core CORDIC entregue un nuevo valor en sus salidas.

- En vez de asignar al core CORDIC una señal de reloj proveniente o derivada de la tarjeta DE2, enviar pulsos desde una botonera (pushbutton

switch), de tal manera que nosotros podemos contar la cantidad de pulsos que recibe el core hasta que se presenten nuevos resultados en sus salidas.

PROCEDIMIENTO PARA EL CONTEO DE FLANCOS

Como se explicó en la sección 3.8 acerca del cálculo de los valores de las funciones Seno y Coseno, el CORDIC Core recibe como valor Ain[15..0] un dato procesado que corresponde al valor de ángulo a calcular.

Para 30°-> Ain[15..0] = 1555 (hex) = 0001 0101 0101 0101 (bin)

Y el resultado que devuelve el core para un valor de entrada Ain=1555hex ya es conocido:

Sen= 16380(dec)=3FFC(hex) Cos= 28381(dec)=6EDD(hex)

De tal modo que usaremos estos datos ya verificados anteriormente para realizar el conteo del número de flancos del modo como sigue:

1. Deshabilitamos el CORDIC Core.

   Switch 17 = 0 (señal habilitadora=0)

2. Colocamos el valor de entrada.

   Switch[15..0] = 0001 0101 0101 0101 (bin). (Correspondiente a un valor de 30 grados como dato de entrada al Core)

3. Luego se habilita el core (Switch17=1) y se realiza el conteo de flancos (o análisis de señales en el caso de usar Quartus II) hasta que en los leds rojos se aprecie el siguiente valor de salida:

   Leds[15..0] = 6EDD hex = 0110 1110 1101 1101

   Correspondiente al resultado del coseno de 30)

4.2.1 SIMULACION DEL CORE CORDIC.

Creamos un nuevo archivo tipo Waveform (solo en versiones de Quartus 9.1 o inferiores) en nuestro proyecto, indicamos al programa las señales que queremos incluir en el proceso de simulación y realizamos el proceso.

Se realizaron simulaciones para los siguientes valores de entrada al Core:

Ángulo de entrada	Ain[15..0]	Sin[15..0]	Cos[15..0]
0	0000000000000000	0000000111001100	1000000000000000
30	0001010101010101	0011111111111111	0110111011011101
60	0010101010101011	0110111011011101	0011111111111111
90	0100000000000000	1000000000000000	0000000111001100

Tabla 4-1 Tabla de valores en binarios obtenidos dado el ángulo de entrada.

En los diagramas mostrados a continuación se presenta una flecha roja que indica el primer flanco que recibe el CORE luego de activarse la señal habilitadora. También se colocó una línea azul entrecortada que indica el último flanco necesario hasta que el core entrega en sus salidas el resultado de Cos y Sin de acuerdo a la tabla anterior.

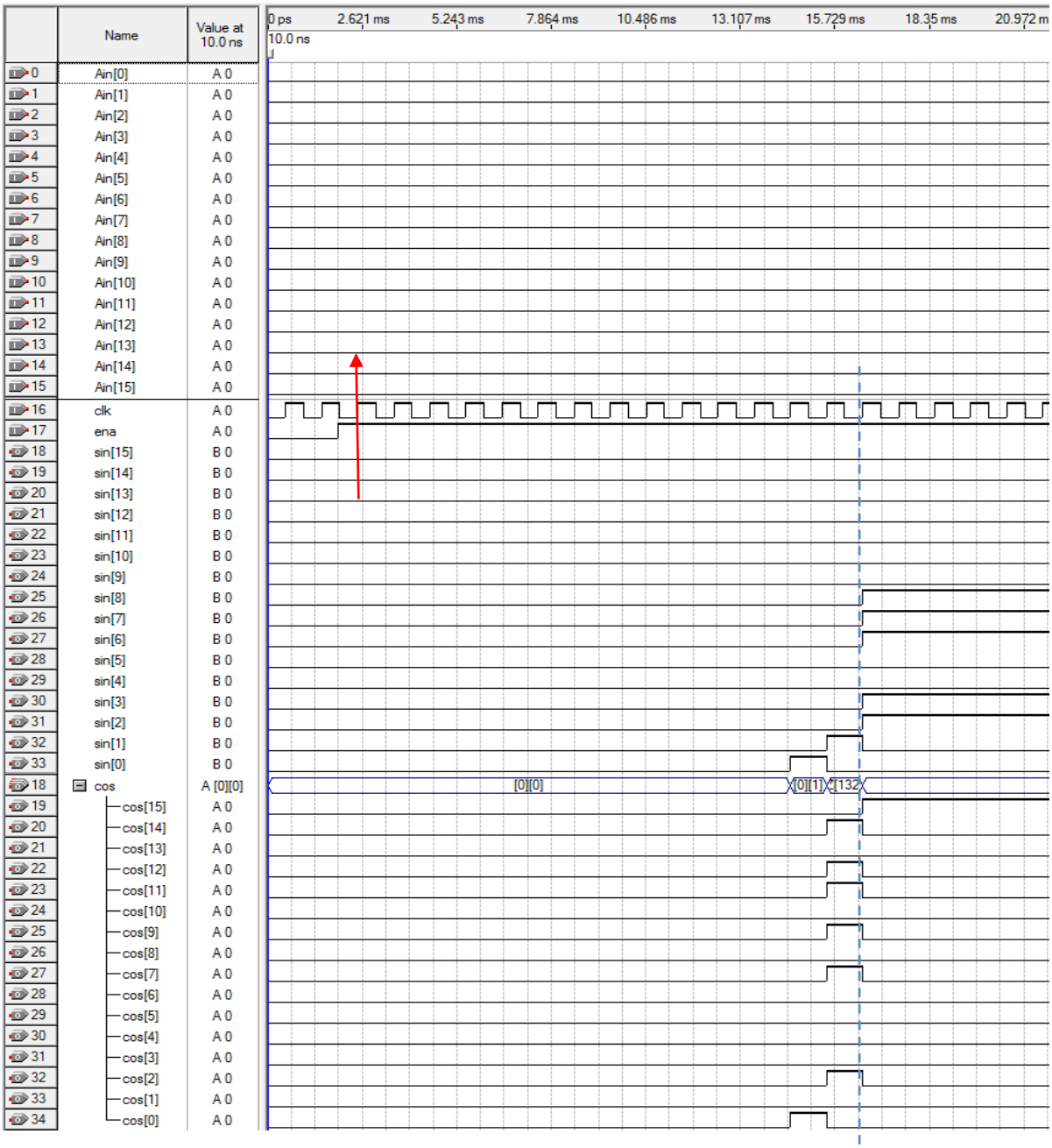


Figura 4-3 Diagrama de tiempo para un ángulo de ingreso al core de 0°

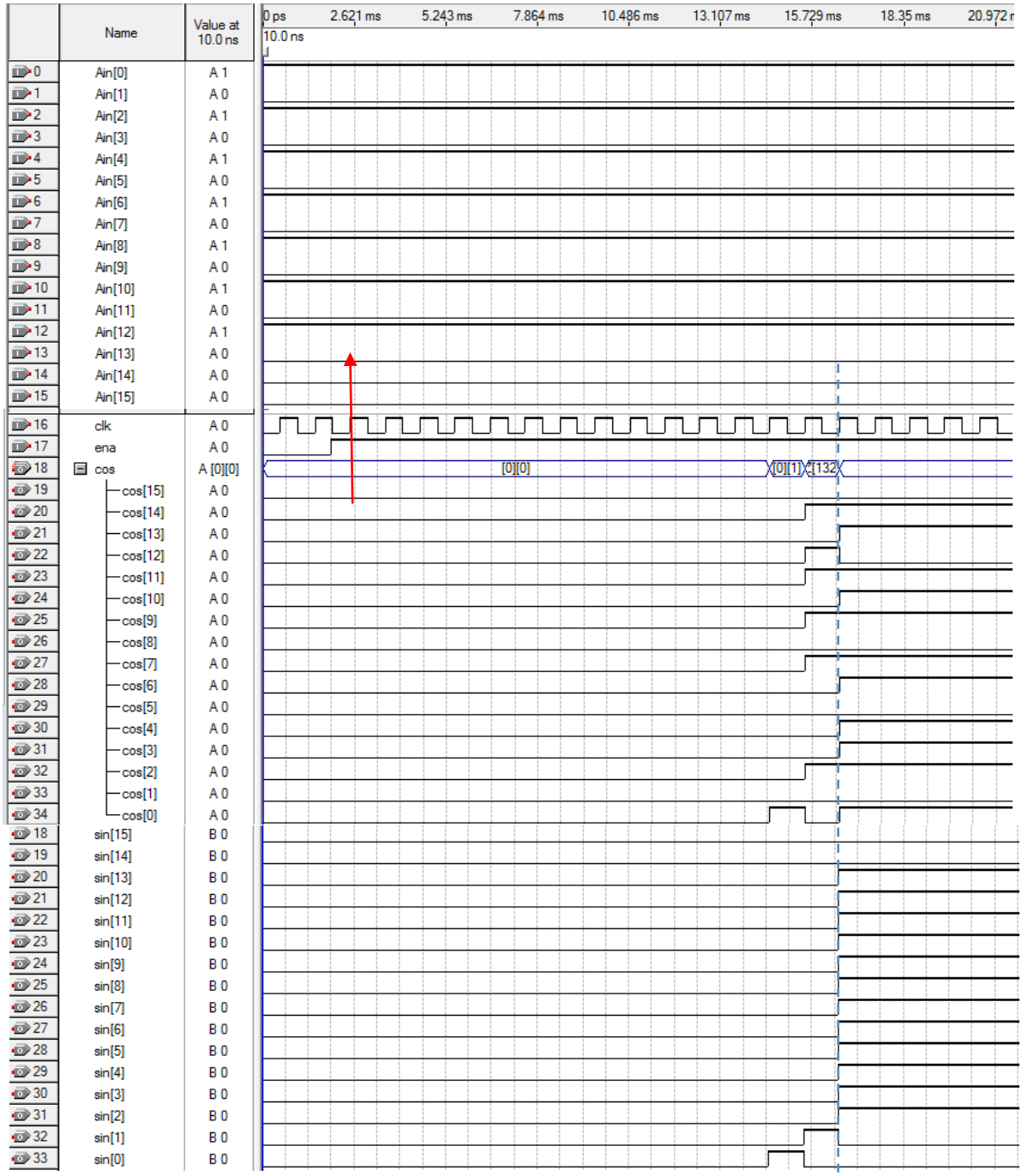


Figura 4-4 Diagrama de tiempo para un ángulo de ingreso al core de 30°

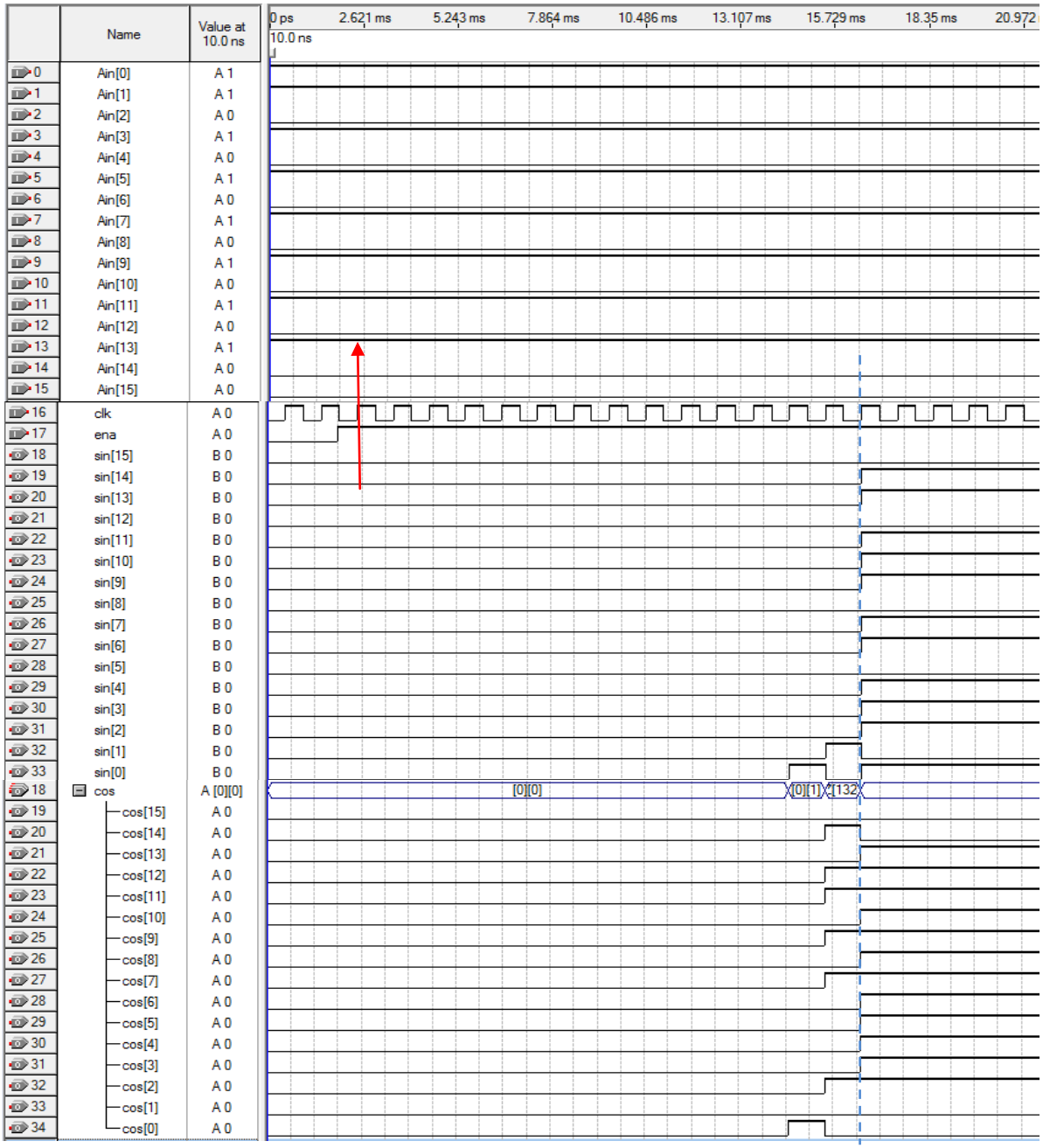


Figura 4-5 Diagrama de tiempo para un ángulo de ingreso al core de 60°

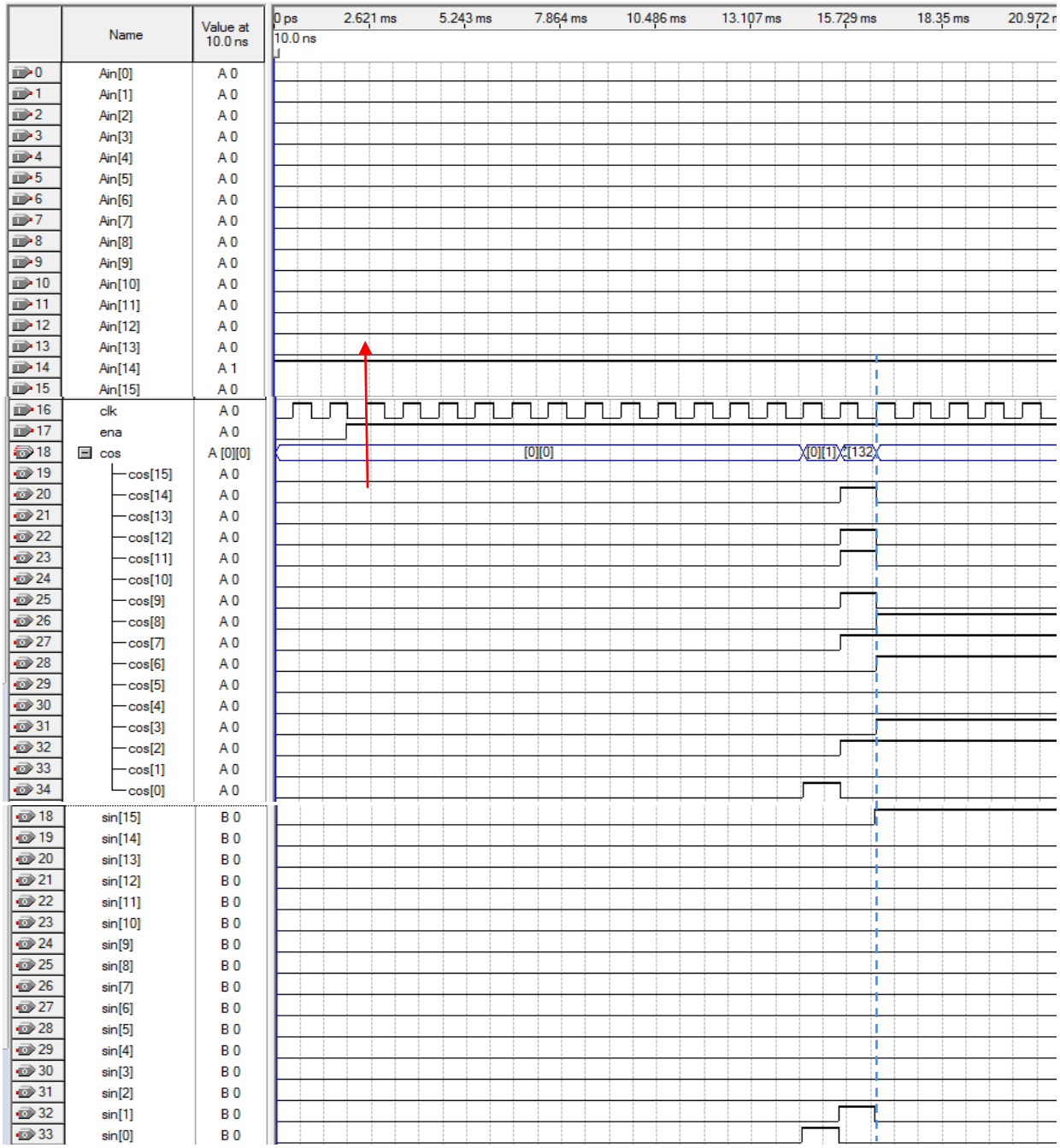


Figura 4-6 Diagrama de tiempo para un ángulo de ingreso al core de 90°

Se puede notar en los diagramas de tiempo mostrados que el core CORDIC entrega el valor correspondiente al resultado (de acuerdo a la Figura 4-6) en el 15avo flanco de reloj que recibe luego de activar y mantener activa la señal de habilitación, sin embargo, entre el 13avo y 14avo flanco se observan otros valores a la salida del core que podrían corresponder a otro valores de pre síntesis usados por el core antes de devolver el resultado final.

4.2.2 ENVIAR PULSOS EXTERNOS AL CORE

En este método, la señal de reloj que recibe el Core se leerá desde una botonera de la tarjeta DE2, cada vez que esta es pulsada indica un flanco de reloj. Para esto realizamos una asignación de pin de la señal de reloj del core con cualquiera de los pushbuttons de la tarjeta, en nuestro caso el Key3.

De igual manera que en el método anterior primero colocamos el valor de Ain en los slides, habilitamos el core (SWITCH_17=1) y empezamos a enviar pulsos al core pulsando el pushbutton key3 (contaremos la cantidad de pulsos que vayamos enviando). Cuando observemos en los leds rojos el valor correspondiente al resultado entonces ya no enviamos

más pulsos y sabremos la cantidad de flancos que necesito el Core para procesar un dato.

El resultado de este experimento fue 15 pulsaciones o como se dijo en los métodos anteriores: 15 flancos.

4.2.3 USO DE CONTADOR PARA CONTEO DE FLANCOS

Debido a que la velocidad de reloj que la tarjeta DE2 puede proveer es de 50MHZ y 27MHZ, vamos a añadir a nuestro esquemático un bloque llamado CLOCK_DIV, el cual divide una señal reloj de 50MHZ en otras señales con menor frecuencia, tal como se muestra a continuación:

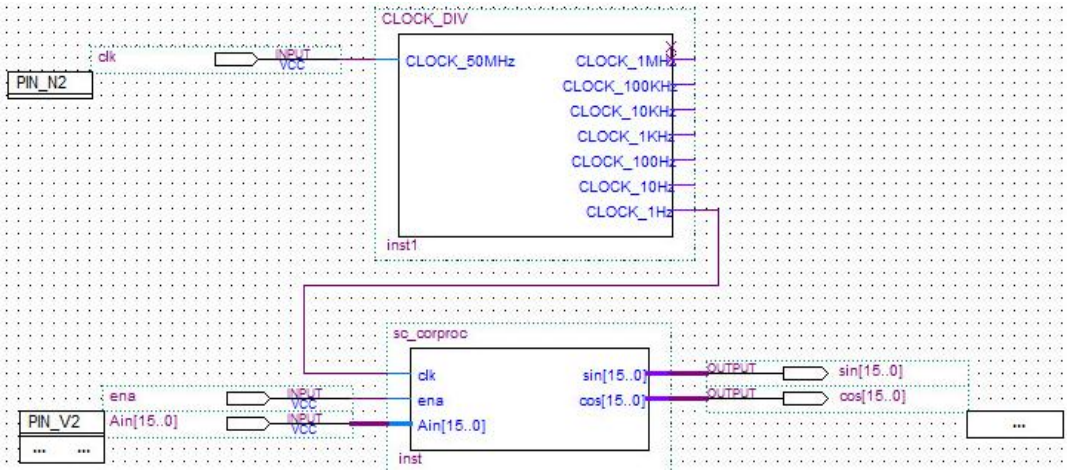


Figura 4-7 Diagrama esquemático de divisor de frecuencia conectado al core para asignarle 1HZ.

Con esta implementación, podemos enviar al core una señal de reloj con frecuencia de 1HZ (1 flanco de subida de reloj cada segundo), para que el ojo humano pueda percibir los cambios que van ocurriendo en los leds rojos que son los que muestran el resultado y la final poder leer el resultado del contador que es mostrado en los leds verdes, si trabajáramos con los 50 MHZ los cambios no serían perceptibles, porque serían demasiado rápido.

Ahora incluiremos un **contador** a nuestro diseño que cumpla las siguientes condiciones:

- Que la señal que habilita el Core desde Switch17 también sea la señal que habilita el contador, de tal modo que cuando el Core empieza a funcionar, el contador empiece su conteo ascendente.

- Tanto el core como el contador tengan la misma señal de reloj (1 Hz).

- El resultado del contador podemos asignarlo a leds verdes (veremos el número en notación binaria)

- Cuando se obtiene el valor de salida del core, lo deshabilitamos (Switch17=0), con lo que el contador también se detendrá y leemos el resultado del contador.

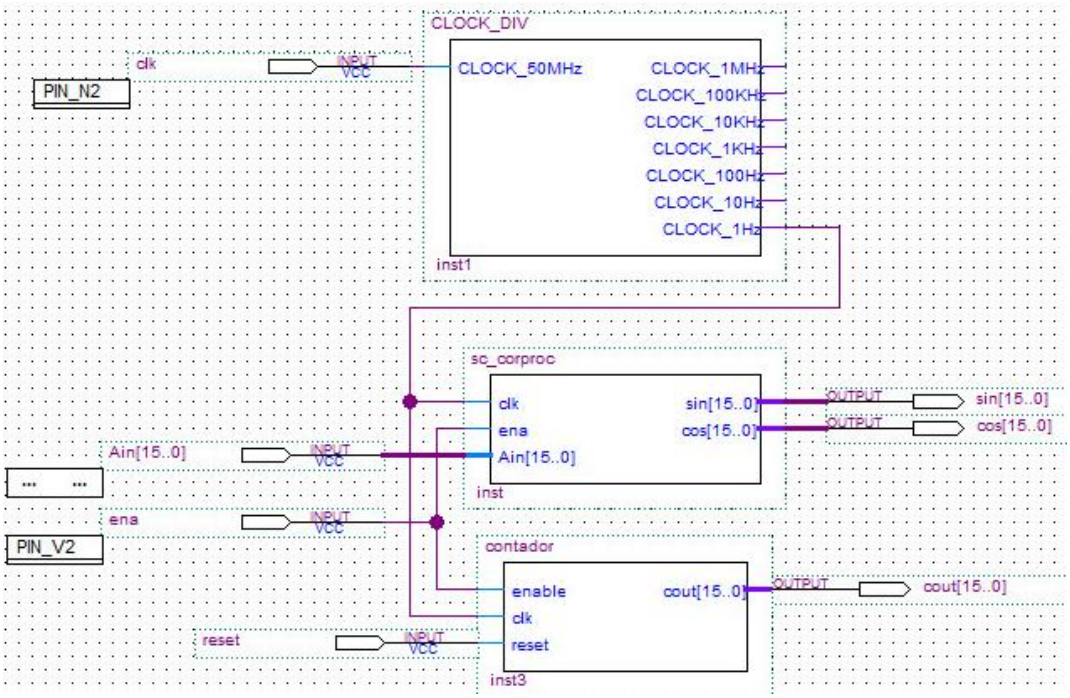


Figura 4-8 Diagrama esquemático del sistema para contar los flancos mediante un bloque contador

Con dichos cambios, nos aseguramos que ambos bloques (contador y CORDIC) empiecen a trabajar en el mismo flanco de reloj, obteniendo los siguientes resultados:

Ángulo X (°)	Ain(X) (dec)	Ain(X) (bin)	Pre-Cos(X) obtenido (bin)	Pre-Cos(X) obtenido (dec)	Cos(X) obtenido (dec)	Tiempo obtenido (Seg)
0	0	0	1000000000000000	32768	1	15
5	910	1110001110	111111110000001	32641	0.99612	15
10	1820	11100011100	111111000001100	32268	0.98474	15
15	2731	101010101011	111101110100101	31653	0.96597	15
20	3641	111000111001	0111100001000100	30788	0.93957	15
25	4551	1000111000111	0111010000000101	29701	0.90640	15
30	5461	1010101010101	0110111011011101	28381	0.86612	15
35	6372	1100011100100	1101000110110 10	26842	0.81915	15
40	7282	1110001110010	110001000001111	25103	0.76608	15
45	8192	10000000000000	0101101010000011	23171	0.70712	15
50	9102	10001110001110	0101001001001000	21064	0.64282	15
55	10012	10011100011100	0100100101101011	18795	0.57357	15
60	10923	10101010101011	0011111111111111	16383	0.49997	15
65	11833	10111000111001	0011011000010101	13845	0.42251	15
70	12743	11000111000111	0010101111000111	11207	0.34201	15
75	13653	11010101010101	0010000100011111	8479	0.25875	15
80	14564	11100011100100	0001011000111011	5691	0.17367	15
85	15474	11110001110010	0000101100101010	2858	0.08722	15
90	16384	100000000000000	0000000111001100	460	0.01403	15

Tabla 4-2 Tabla de resultados obtenidos mediante el sistema de la Figura 4-8 con periodo de reloj 1Hz

Luego de observar los resultados de la tabla anterior, podemos decir con más firmeza que el número de flancos que requiere el CORDIC Core

hasta converger es de **15 flancos, valor que ya fue obtenido anteriormente en la simulación del core**.

4.3 IMPLEMENTACION DEL ALGORITMO CORDIC ESCRITO EN C

Para saber que tan eficiente es el core CORDIC, realizamos otra implementación pero esta vez usamos el algoritmo CORDIC en Lenguaje C y le medimos el tiempo que toma en procesar un dato.

En la web es posible encontrar varias implementaciones del algoritmo CORDIC publicados por otros desarrolladores, desde códigos que operan en la misma función principal MAIN() del programa hasta implementaciones donde se incluyen librerías y funciones adicionales para su procesamiento. Sin embargo, recordemos que nuestro CORE fue diseñado con una arquitectura en paralelo, con 16 bloques internos que procesan los datos correspondientes a 16 iteraciones del algoritmo, por lo que debemos compararlo con una implementación en C del algoritmo que realice también 16 iteraciones (existen implementaciones del algoritmo para 16 y 32 iteraciones, mientras más iteraciones se realicen la precisión del resultado mejora pero se requiere mayor espacio en memoria para procesamiento de datos) y que además sea lo más sencilla posible, además, el código que utilicemos lo debemos ejecutar en el mismo hardware donde se ejecutó el

core CORDIC de tal modo que la comparación entre el uso del core o implementación en lenguaje C sea justa.

4.3.1 CODIGO A UTILIZAR EN LAS PRUEBAS

Luego de analizar varias implementaciones se eligió utilizar la siguiente, debido a que la forma de procesar el ángulo de entrada (para obtener el valor de Z) es muy parecido al proceso realizado por el Core Cordic y además esta implementación en lenguaje C tiene una precisión de 16 iteraciones similar a la utilizada por el Core.

```
#include <stdio.h>
#define cordic_1K 0x000026DD
#define to_radianes 0.017453292
#define MUL 16384.000000
#define CORDIC_NTAB 16
int cordic_ctab [] = {0x00003243, 0x00001DAC, 0x00000FAD, 0x000007F5,
0x000003FE, 0x000001FF, 0x000000FF, 0x0000007F, 0x0000003F,
0x0000001F, 0x0000000F, 0x00000007, 0x00000003, 0x00000001,
0x00000000, 0x00000000 };

int main(){
                int t1=0,t2=0,t3=0,p1,p2;
                int angulo1=0, seno, coseno;
                int angulo2=angulo1*to_radianes*MUL;

                  int k, d, tx, ty, tz;
                  int x=cordic_1K,y=0,z=angulo2;

        for (k=0; k<CORDIC_NTAB; ++k)
                {
                        d = z>>(CORDIC_NTAB-1);
                        tx = x - (((y>>k) ^ d) - d);
                        ty = y + (((x>>k) ^ d) - d);
                        tz = z - ((cordic_ctab[k] ^ d) - d);
                        x = tx; y = ty; z = tz;
                }
                coseno = x/MUL; seno = y/MUL;
        return 0;
}
```

En las primeras líneas se definen algunas constantes a utilizar por el algoritmo, se define un vector cordic_ctab[] como looktable que contiene valores de arcotangente ya conocidos necesarios para el algoritmo, y además se define CORDIC_NTAB igual a 16 que indica el número de iteraciones a realizarse.

Esta implementación del algoritmo requiere que primero se realice una modificación al valor de ángulo a calcular (así como también hicimos al enviar un dato de entrada al core CORDIC):

```
int angulo2=angulo1*to_radianes*MUL;
```

Las constantes **to_radianes** y **MUL** representan el valor de PI/180 y 2^{14} respectivamente. Esto es necesario para que el algoritmo pueda realizar los desplazamientos y sumas necesarios sobre los valores.

En la función principal también puede apreciarse una estructura **FOR** que realizará 16 iteraciones, donde en cada una de ellas varios desplazamientos, sumas, restas o búsquedas acumularan los valores respectivos de seno y coseno en variables temporales tx y ty, el valor de ángulo restante por reducir se almacena en tz, cuyo valor inicial es el ángulo a calcular. Cuando las iteraciones van terminando, el valor de tz se aproxima a cero y los valores de seno y coseno se colocan en las

variables X y Y (si se realizaran 32 iteraciones el valor de TZ obtenido al final se aproximaría mucho más a cero y los valores de seno y coseno serían mucho más precisos aunque no los exactos).

Los valores de X y Y obtenidos al final de las iteraciones deben ser divididos para el valor de MUL con lo que se obtienen los valores deseados en base decimal para seno y coseno.

Cabe recalcar que el proceso que realiza el algoritmo CORDIC es únicamente el mostrado en las iteraciones del lazo **FOR**, el resto del código escrito para preparar el ángulo y al final del código para obtener el valor final de ángulo son herramientas para completar las pruebas satisfactoriamente.

Únicamente se medirá el tiempo que toman las iteraciones en acumular los valores de tx y ty cuando tz se aproxima a cero una vez que las iteraciones terminan.

```
#include <stdio.h>
#include <sys/alt_alarm.h>
#include <sys/time.h>
#include <sys/alt_timestamp.h>
#define cordic_1K 0x000026DD
#define to_radianes 0.017453292
#define MUL 16384.000000
#define CORDIC_NTAB 16
int cordic_ctab [] = {0x00003243, 0x00001DAC, 0x00000FAD,
0x000007F5, 0x000003FE, 0x000001FF, 0x000000FF, 0x0000007F,
0x0000003F, 0x0000001F, 0x0000000F, 0x00000007, 0x00000003,
0x00000001, 0x00000000, 0x00000000 };
```

```
int main(){
        while(1){
                int rr=alt_ticks_per_second();
                int t1=0,t2=0,t3=0,p1,p2;
                int angulo1=0, seno, coseno;
                printf("Ingrese angulo: ");
                scanf("%d",&angulo1);
                int angulo2=angulo1*to_radianes*MUL;

                  int k, d, tx, ty, tz;
                  int x=cordic_1K,y=0,z=angulo2;
                  t1=alt_timestamp_start();
                  for (k=0; k<CORDIC_NTAB; ++k)
                  {
                          d = z>>(CORDIC_NTAB-1);
                          tx = x - (((y>>k) ^ d) - d);
                          ty = y + (((x>>k) ^ d) - d);
                          tz = z - ((cordic_ctab[k] ^ d) - d);
                          x = tx; y = ty; z = tz;
                  }
                  t2=alt_timestamp();
                  coseno = x; seno = y;

        printf("Sen(%d)=%f;Cos(%d)=%fn",angulo1,seno/MUL,angulo1,coseno/MUL);
        printf("Tiempo de cálculo de algoritmo t=%d \n",t2);
        }
        return 0;
}
```

En nuestra implementación se añadieron nuevas líneas de código para pedir al usuario el ángulo de ingreso. Además se añadieron las funciones: alt_time_start() y alt_timestamp(). La primera de ellas inicia el TimeStamp Timer desde cero (un contador del que hablaremos a continuación) y la segunda obtiene el valor actual del contador ya mencionado.

De acuerdo a como hemos colocado las dos nuevas funciones en el código hacemos entender que se inicia el contador (alt_time_start()) justo antes de empezar las iteraciones, y obtenemos el valor del contador

(alt_timestamp()) justo luego de terminar la última iteración, con lo que el valor obtenido del contador es el número aproximado de flancos que tomó el procesador Nios II embebido en procesar las iteraciones.

4.3.2 TIMESTAMP TIMER

El TimeStamp es un componente útil para medir el rendimiento de un sistema NiosII [2]. Para su uso es necesario añadir a nuestro hardware un nuevo core **Timer** con la herramienta SOPC Builder, generar la nueva máquina y compilar nuevamente nuestro proyecto en Quartus II**.** Nuestro sistema en realidad ya contaba con un timer pero que es usado como SYS_CLK_TIMER en nuestro sistema NIOS II, y para hacer uso del TIMESTAMP debemos utilizar un **Timer** diferente.

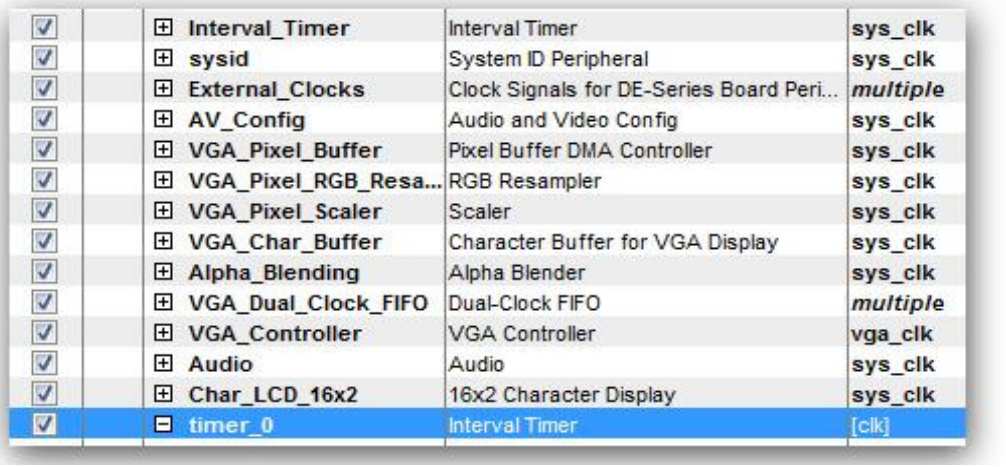


Figura 4-9 Agregando el timer 0 en SopcBuilder

Una vez que se compiló el proyecto en QuartusII con el nuevo **Timer** incluido, debemos indicar en el BSP Editor (en nuestro proyecto de NIOS II SBT, damos clic derecho y vamos hasta Nios II -> BSP Editor) el Timer que se usará como Time Stamp Timer. Generamos el archivo de nuestra nueva configuración y compilamos nuestro proyecto en Nios II IDE.

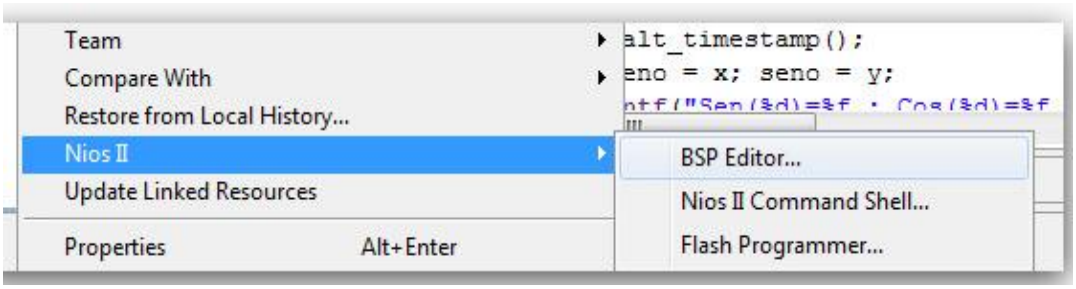


Figura 4-10 Ingresar a BSP Editor

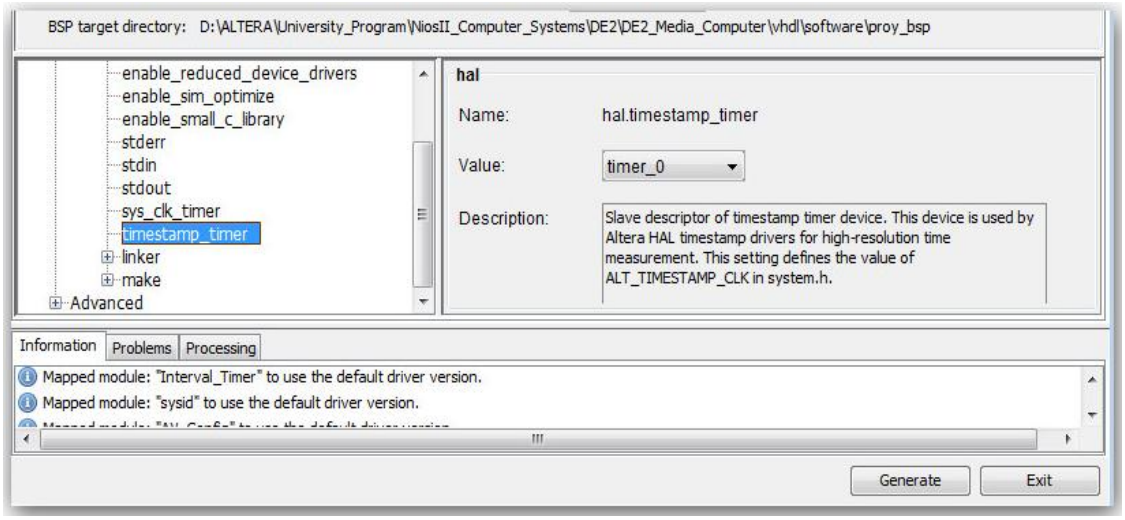


Figura 4-11 Configurando el timer 0.

Una vez incluido un nuevo timer en nuestro hardware, y haber indicado en el BSP editor el timestamp timer ya podemos hacer uso de las funciones alt_time_start() y alt_timestamp(). Cabe indicar que el timer usado para TimeStamp es alimentado por una señal de reloj de 50Mhz, por lo que el valor que retorne la función alt_timestamp() será en unidades de flancos de dicha velocidad de reloj.

4.3.3 PRUEBA DE RENDIMIENTO DEL ALGORITMO CORDIC IMPLEMENTADO EN LENGUAJE C

El análisis del código se realiza de corrido, no es posible realizar un análisis de rendimiento con las herramientas de debugging ya que ellas hacen interrupciones en la ejecución de nuestro código, aumentando el número de flancos de la ejecución del programa.

Cuando se ejecuta nuestro código el usuario ingresa el valor de ángulo por teclado, el algoritmo procesa el dato y las funciones de timestamp nos ayudan a medir la cantidad de flancos utilizados, además, los datos de seno, coseno y el resultado de la función timestamp() se muestran en pantalla. Como todo el código se encuentra dentro de una estructura WHILE, luego de que se calcula el seno y coseno de un ángulo, se nos

pedirá ingresar un nuevo valor, realizando así pruebas para varios valores de ángulos.

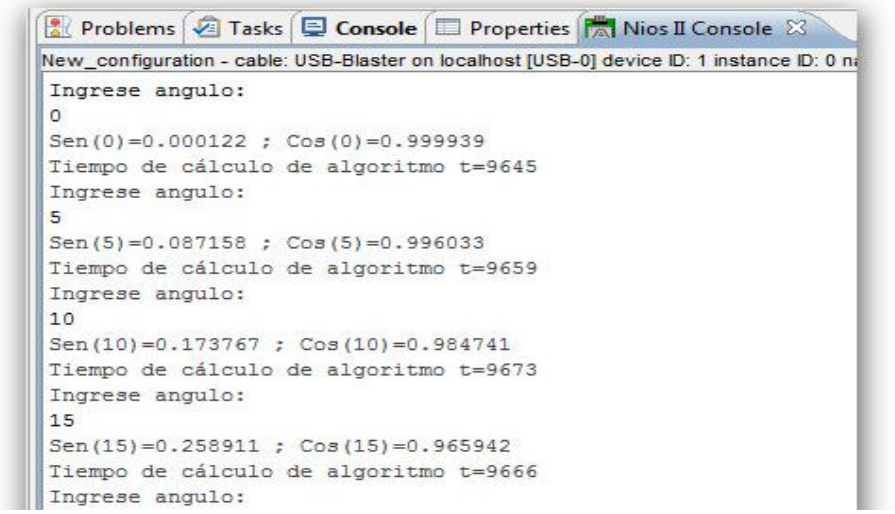


Figura 4-12 Ejecución del código CORDIC en lenguaje C.

La tabla a continuación muestra la cantidad de flancos que se necesitaron para procesar el algoritmo CORDIC sobre algunos valores de ángulos.

ÁNGULO	PRUEBA 1	PRUEBA 2	PRUEBA 3	PRUEBA 4	PRUEBA 5	SUB PROMEDIO
0	9645	9683	9665	9668	9662	9664,6
5	9673	9683	9682	9698	9679	9683
10	9662	9678	9656	9678	9666	9668
15	9674	9699	9681	9662	9690	9681,2
20	9688	9698	9656	9676	9674	9678,4

25	9672	9674	9679	9670	9680	9675
30	9675	9684	9678	9666	9678	9676,2
35	9689	9636	9661	9673	9670	9665,8
40	9683	9672	9656	9673	9673	9671,4
45	9674	9688	9680	9674	9656	9674,4
50	9674	9653	9668	9666	9670	9666,2
55	9661	9665	9680	9674	9680	9672
60	9673	9672	9665	9665	9699	9674,8
65	9646	9680	9656	9645	9656	9656,6
70	9673	9678	9661	9667	9673	9670,4
75	9646	9665	9683	9684	9684	9672,4
80	9668	9668	9670	9687	9673	9673,2
85	9689	9683	9665	9672	9661	9674
90	9680	9668	9671	9654	9672	9669

Tabla 4-3 Tabla de cantidad de flancos obtenidos en las diferente pruebas.

Donde el número de flancos en promedio que requiere el algoritmo CORDIC implementado en C es de 9671, para terminar las 16 iteraciones.

4.4 ANALISIS DE RESULTADOS

La siguiente tabla resume los resultados obtenidos en las pruebas realizadas:

PRUEBA	NUMERO DE FLANCOS UTILIZADOS
Simulación Core CORDIC	15
Contador de flancos – Core CORDIC	15
Envío de pulsos – Core CORDIC	15
IMPLEMENTACION EN C	9671 (aprox)

Tabla 4-4 Tabla de resultados obtenido en las diferente pruebas.

El número de flancos que requiere el core CORDIC (15 flancos) para procesar un dato es independiente de la frecuencia de reloj a la que éste trabaje, y en nuestro sistema NIOS II el core CORDIC trabaja a 50 MHZ, la misma frecuencia utilizada por el core TIMER asignado al TimeStamp Timer, entonces en términos de tiempos podemos asegurar:

	Core CORDIC	Implementación en C
Número de flancos requeridos	15	9671
Frecuencia	50MHZ	50MHZ
Tiempo aproximado de procesamiento	O.3 useg	193.42 useg

Tabla 4-5 Comparación de la medición de tiempo de respuesta en diferentes escenarios.

Donde el core CORDIC retorna en sus salidas el resultado 644.73 veces más rápido que su implementación en C.

Esta diferencia se debe porque que el core CORDIC recibe el dato a calcular de manera directa (no pasa por el procesador NIOS), lo procesa y arroja una respuesta, pero al usar el algoritmo CORDIC en lenguaje C, para procesar un dato el código tiene que ser cargado en memoria y luego al procesador que es donde se va a ejecutar el código, lo que toma mas tiempo obtener una respuesta a diferencia del core CORDIC.

CONCLUSIONES

1. El desarrollo de sistemas embebidos posee enorme versatilidad frente a cambios en el diseño de hardware o software, ya que pueden ser modificados en cualquier momento sin alterar o añadir hardware extra en el PBC a diferencia de los métodos tradicionales.

2. La ventaja del uso del algoritmo CORDIC radica en la posibilidad de realizar varias funciones matemáticas (descritas en el capítulo 2) en hardware que no requiere multiplicadores o unidades aritméticas complejas debido a que el algoritmo se resuelve en base a operaciones de desplazamientos, búsquedas y sumas y restas.

3. El algoritmo CORDIC puede ejecutarse con el número de iteraciones que el programador elija, recordando que mientras el número de iteraciones aumenta la precisión del resultado mejora, pero con menor número de iteraciones la velocidad del cálculo se agilita.

4. El uso de canalización (pipeline) de los bloques internos en la arquitectura del Core CORDIC reduce el tiempo de propagación del bit de acarreo de los sumadores o restadores, haciendo a éste el diseño más eficiente para su implementación en lenguajes HDL.

5. El uso de un Core para realizar una función específica dentro de un sistema embebido, proporciona respuestas a funciones mucho más rápido que si implementáramos dicha tarea en el código de una aplicación que corre sobre un microprocesador. Gracias a esta ventaja, el uso de varios cores funcionando como bloques de un sistema distribuido brinda al programador respuestas de proceso casi en tiempo real. Esta característica es muy importante en procesos donde se requieren respuestas rápidas de funciones trigonométricas para toma de decisiones tales como en la aviación, procesamiento de señales, procesamiento de imágenes y otros.

6. Aunque la implementación de un Core en nuestro diseño de hardware ocupa más celdas lógicas del FPGA donde se implementa, el tiempo de procesamiento de la función que realizará el Core se reduce considerablemente.

7. El trabajo de este proyecto fue únicamente con un Core que entrega resultados de funciones senos y cosenos, sin embargo puede desarrollarse un nuevo proyecto de sistema embebido para funciones hiperbólicas, logarítmicas y otras funciones trascendentales, idéntico al mostrado en este documento únicamente, modificando el Core CORDIC implementado.

RECOMENDACIONES

1. Si se está usando la tarjeta DE2 de Altera, se aconseja revisar los foros de altera para solución de problemas que se puedan presentar.

2. Se recomienda usar la computadora media o básica que brinda SPOC Builder y desde allí agregar o eliminar los módulos de acuerdo a las exigencias del desarrollador.

3. Sería conveniente trabajar con un computador con características medias del hardware (preferiblemente procesadores core 2 duo o core i series en adelante) para el desarrollo de este tipo de proyectos ya que la generación y compilación de tal, requiere de recursos considerables del computador para su agilidad.

4. Se recomienda que la interconexión del Core con el sistema creado con SOPC Builder se realice modificando el código del sistema embebido que contiene a los dos entidades anteriores. No es posible interconectar el core

con el sistema en archivos esquemáticos. Además para realizar la respectiva conexión entre ellos se debe agregar al sistema los respectivos PIO (Puertos de entrada y salida), que intercambiarán datos con el core.

5. Para enviar señales externas por medio de los puertos de expansión JPE2, se recomienda utilizar resistencias en cada uno de los pines a usar, para proteger a la tarjeta de cambios drásticos de voltaje. Los pines del puerto JP2 pueden receptar voltajes de 3.3 a 5v.

6. Se sugiere por facilidad el uso de TIMESTAMP Driver para el análisis del rendimiento del software. Existe muchos otros métodos basados en uso de consola de NIOS II que no son parte de este trabajo pero que detallan más información sobre rendimiento.

7. Para analizar el comportamiento en las lecturas de señales desde el teclado externo hacia la tarjeta se recomienda utilizar las herramientas de debugging proporcionadas por NIOS II SBT, que facilitaron la resolución de varios inconvenientes en el desarrollo del software.

ANEXOS

ANEXO A

CODIGO FUENTE EN LENGUAJE C

```
/*********************************************************************************
 * CALCULADORA DE FUNCIONES TRASCEDENTALES BASADA EN EL ALGORITMO CORDIC
*********************************************************************************/
// LIBRERIAS INCLUIDAS
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <io.h>
#include "system.h"
#include "altera_up_avalon_parallel_port.h"
#include "altera_avalon_pio_regs.h"
#include "altera_up_avalon_character_lcd.h"
#include "altera_up_avalon_character_lcd_regs.h"
#include <ctype.h>
#include <math.h>
#include <time.h>

// DEFINICION DE CONSTANTES QUE REPRESENTAN LA DIRECCION DE MEMORIA DE ALGUNOS MÓDULOS
// DE LA TARJETA DE2
#define leg      0x10000010                    // LEDS VERDES
#define gpio_in0x00800000                      // PIO ENTRADA        in
#define gpio_out         0x00800010            // PIO SALIDA         out
#defineintToChar(entero) entero+0x30
#define zeta     0x00800040                    // PIO ZETA           out
#define sen      0x00800020                    // PIO SENO           in
#define cos      0x00800030                    // PIO COSENO         out

// FUNCIONES IMPLEMENTADAS
int esdigito(char tec);
void lcd_clear_screen(alt_up_character_lcd_dev *lcd,int len,int delay1, int delay2);
char * convertIntChar(int num);
char leer_tecla();
int char2int(char c);
int string2int( char c[]);
char* dispTop(char fun,char signo,char numero[5],int digi);
char* dispBot(float res,char signo,int cuadrante);
int to_angulo_menor90(int angulo);
int Get_cuadrante(int angulo);
float resultado_funcion(int num_bin);
int binario_to_decimal(char resultado_binario[]);
```

```
char get_signo(int cuadro, char tecla);
int Get_cuadrante2(int angulo, char signo_in);

// FUNCION PRINCIPAL
int main(){
        //PUNTEROS UTILIZADOS
        volatile long *p_zeta=zeta;
        volatile long *pleg=leg;
        volatile int *psen=sen;
        volatile int *pcos=cos;
        alt_up_character_lcd_dev * lcd;
        lcd = alt_up_character_lcd_open_dev(CHAR_LCD_16X2_NAME);
        //ENTEROS
        int valorsen=0,valorcos=0,dig=0,cuadrante,len=15,x=0;
   int ingresar=0, borrar=0, j=0,totalint=0,reducido=0;long total=0;

        //CARACTERES Y CADENAS
        char tecla='q';             // valor default de tecla
        char num[5]="qqqqq";   // cadena de caracteres donde se almacenaran los digitos
                                   ingresados que se manejaran en el programa
        char num2[5]="     ";      // cadena de caracteres donde se almacenaran los digitos
                                ingresados que se mostraran en pantalla
        char sig='a';               // signo positivo (a) por defecto de un numero. r (negativo)
        char funcion='q';           // valor default de funcion
        char *linea1="";
        char *linea2="";
        char signo;

        // MENSAJE DE INICIO EN LCD
        alt_up_character_lcd_set_cursor_pos(lcd, 0, 0);
   alt_up_character_lcd_write(lcd," <CALCULADORA> ", len);

   alt_up_character_lcd_set_cursor_pos(lcd, 0, 1);
   alt_up_character_lcd_write(lcd,"  Sen&Cos&Tan  ", len);

   do{
   }while(leer_tecla()!='e');

   while(1){
   *pleg=0;
      while(1){
         if(x==0)
         {
                  alt_up_character_lcd_set_cursor_pos(lcd, 0, 1);
                  alt_up_character_lcd_write(lcd,"Valor:         ", len);
                  alt_up_character_lcd_set_cursor_pos(lcd, 0, 0);
                  alt_up_character_lcd_write(lcd,"Funcion:       ", len);
                  alt_up_character_lcd_set_cursor_pos(lcd, 0, 0);
                  alt_up_character_lcd_write(lcd,"Funcion: ", 9);
```

```
        }
        tecla=leer_tecla();
        if(tecla=='s' || tecla=='c' || tecla=='t' || tecla=='e'){
                if(tecla=='s')
                {
                //************************ FUNCION        SENO *****************************
                        alt_up_character_lcd_set_cursor_pos(lcd, 0, 0);
                        alt_up_character_lcd_write(lcd,"Funcion: Sen() ", len);
                        funcion=tecla;
                x=1;
                }else if(tecla=='c')
                {
                        //************************ FUNCION COSENO*****************************
                        alt_up_character_lcd_set_cursor_pos(lcd, 0, 0);
                        alt_up_character_lcd_write(lcd,"Funcion: Cos() ", len);
                        funcion=tecla;
                x=1;
                }else if(tecla=='t')
                {
                        //************************ FUNCION TANGENTE ****************************
                        alt_up_character_lcd_set_cursor_pos(lcd, 0, 0);
                        alt_up_character_lcd_write(lcd,"Funcion: Tan() ", len);
                        funcion=tecla;
                x=1;
           }
                else if(tecla=='e'){
                        if(x==1)
                                x=2;
                }
        }
        if(x==2)
                break;
    }//-> Hasta aquí, última revisión el 18 mayo 2013

  // FUNCION INGRESADA
    *pleg=1;    // primera señal
    ingresar=1;
    borrar=0;
    alt_up_character_lcd_set_cursor_pos(lcd, 0, 1);
    alt_up_character_lcd_write(lcd,"Valor:         ", len);
    alt_up_character_lcd_set_cursor_pos(lcd, 0, 1);
    alt_up_character_lcd_write(lcd,"Valor:   ",9);

    while(1)
    {
        tecla=leer_tecla();
        if(esdigito(tecla) && dig<5){       // solo se pueden ingresar hasta 5 digitos
                num[dig]=tecla;
                                dig=dig+1;
```

```
                        for (j=0;j<5;j++){
                                if(num[j]!='q')
                                        num2[j]=num[j];
                        }

                        //Mostrar numero en pantalla
                        alt_up_character_lcd_set_cursor_pos(lcd, 0, 1);
                        alt_up_character_lcd_write(lcd,"Valor:         ", len);
                        if(sig=='a'){
                                alt_up_character_lcd_set_cursor_pos(lcd, 9, 1);
                                alt_up_character_lcd_write(lcd,"+",1);
                        }
                        else if(sig=='r'){
                                alt_up_character_lcd_set_cursor_pos(lcd, 9, 1);
                                alt_up_character_lcd_write(lcd,"-",1);
                        }
                        alt_up_character_lcd_set_cursor_pos(lcd, 10, 1);
                        alt_up_character_lcd_write(lcd,num2,dig);
}else if(tecla=='m'){

        // Cambiar signo
        if(sig=='a')
                        sig='r';
                else if(sig=='r')
                        sig='a';
                //Mostrar numero en pantalla
                alt_up_character_lcd_set_cursor_pos(lcd, 0, 1);
                alt_up_character_lcd_write(lcd,"Valor:         ", len);
                if(sig=='a'){
                        alt_up_character_lcd_set_cursor_pos(lcd, 9, 1);
                        alt_up_character_lcd_write(lcd,"+",1);
                }
                else if(sig=='r'){
                        alt_up_character_lcd_set_cursor_pos(lcd, 9, 1);
                        alt_up_character_lcd_write(lcd,"-",1);
                }
                alt_up_character_lcd_set_cursor_pos(lcd, 10, 1);
                alt_up_character_lcd_write(lcd,num2,dig);
                //num2[5]="     ";
                //Fin mostrar numero en pantalla
}else if(tecla=='l'){
        // Borrar ultimo digito
        if(dig>0){
        num[dig-1]="q";
        num2[dig-1]=" ";
        dig=dig-1;

        if(sig=='a'){
                alt_up_character_lcd_set_cursor_pos(lcd, 9, 1);
```

```
                    alt_up_character_lcd_write(lcd,"+",1);
                }
                else if(sig=='r'){
                    alt_up_character_lcd_set_cursor_pos(lcd, 9, 1);
                    alt_up_character_lcd_write(lcd,"-",1);
                }
                alt_up_character_lcd_set_cursor_pos(lcd, 10, 1);
                alt_up_character_lcd_write(lcd,"     ",5);
                alt_up_character_lcd_set_cursor_pos(lcd, 10, 1);
                alt_up_character_lcd_write(lcd,num2,dig);

                }
    }else if(tecla=='e'){
                total=string2int(num);
                totalint=total;
                if(dig>0)
                    x=10;
    }
    if(x==10)
                break;
 }
//    SEGUNDA SEÑAL //////////////////////
*pleg=3;
//Validar angulo
cuadrante=Get_cuadrante2(total,sig);
printf("Ingreso: %c %d\n",sig,total);
signo=get_signo(cuadrante,funcion);
total=to_angulo_menor90(total);
reducido=total;
printf("angulo reducido: %d, cuadrante: %d, signo: %c \n ",total,cuadrante,signo);

//1. Calcular Z en decimal
total=(total*(pow(2,16)))/360;
total=total+65536; // se le añade el bit de activacion del core
*p_zeta=total;
//*********************************************************************
valorsen=*psen;
//*********************************************************************
valorcos=*pcos;
//*********************************************************************
printf("sen: %d cos: %d",valorsen,valorcos);
float senf,cosf;
senf=resultado_funcion(valorsen);
cosf=resultado_funcion(valorcos);
if(funcion=='s'){
        *linea1=dispTop('s',sig,num2,dig);
}else if(funcion=='c'){
        *linea1=dispTop('c',sig,num2,dig);
}else if(funcion=='t'){
```

```
        *linea1=dispTop('t',sig,num2,dig);
}
alt_up_character_lcd_set_cursor_pos(lcd, 0, 0);
alt_up_character_lcd_write(lcd,linea1, len);
if(funcion=='s'){
        *linea2=dispBot(senf,signo,cuadrante);
        alt_up_character_lcd_set_cursor_pos (lcd, 0, 1);
        alt_up_character_lcd_write(lcd,linea2, len);
}else if(funcion=='c'){
        *linea2=dispBot(cosf,signo,cuadrante);
        alt_up_character_lcd_set_cursor_pos (lcd, 0, 1);
        alt_up_character_lcd_write(lcd,linea2, len);
}else if(funcion=='t'){
        float tagf=senf/cosf;
        while(totalint>=360)
        {
                totalint=totalint-360;
        }
        if(totalint==90 || totalint==270){
                alt_up_character_lcd_set_cursor_pos (lcd, 0, 1);
                alt_up_character_lcd_write(lcd,"Math Error     ", len);
        }
        else if(totalint==0 || totalint==180){
                alt_up_character_lcd_set_cursor_pos (lcd, 0, 1);
                alt_up_character_lcd_write(lcd,"0              ", len);
        }
        else{
                *linea2=dispBot(tagf,signo,cuadrante);
                alt_up_character_lcd_set_cursor_pos (lcd, 0, 1);
                alt_up_character_lcd_write(lcd,linea2, len);
        }
}
*pleg=7;
do{
    tecla=leer_tecla();
}while(tecla!='e');

*pleg=8;
alt_up_character_lcd_set_cursor_pos(lcd, 0, 0);
alt_up_character_lcd_write(lcd, "  CALCULADORA  ", 15);
alt_up_character_lcd_set_cursor_pos (lcd, 0, 1);
alt_up_character_lcd_write(lcd, "  Sen-Cos-Tan  ", len);
do{
}while(leer_tecla()!='e');
x=0;
dig=0;
tecla='q';
for(j=0;j<5;j++)
{
```

```
            num[j]='q';
            num2[j]=' ';
        }
        j=0;
        sig='a';
        signo='+';
        funcion='q';
        *linea1="";
        *linea2="";
        cuadrante=1;
        printf("vuelta!\n");
    }
    return 0;
}
char leer_tecla(){
// Envia y recibe las señales correspondientes al teclado hexadecimal y retorna un CHAR
// que representa a la tecla presionada: 0,1,2,3,4,5,6,7,8,9, m (signo),l (limpiar),
// s (seno), c (coseno),t (tangente),e (enter)

    volatile int *p_gpio_in=gpio_in;
    volatile int *p_gpio_out=gpio_out;
    unsigned char fila[]={0x7,0xb,0xd,0xe};
    int i=0,j=0;
    char tecla='w';
    while(1){
        *p_gpio_out=fila[i];
        if(*p_gpio_in!=15){
            switch (*p_gpio_in)
            {
                case 14:// columna 0
                    switch(i)
                    {
                        case 0:
                            tecla='1'; j=1;
                            break;
                        case 1:
                            tecla='4';j=1;
                            break;
                        case 2:
                            tecla='7';j=1;
                            break;
                        case 3:
                            tecla='m';j=1;// +/-
                            break;
                        default:
                            break;
                    }
                    break;
                case 13:
```

```
                switch(i)
                {
                        case 0:
                                tecla='2';j=1;
                                break;
                        case 1:
                                tecla='5';j=1;
                                break;
                        case 2:
                                tecla='8';j=1;
                                break;
                        case 3:
                                tecla='0';j=1;
                                break;
                        default:
                                break;
                }
                break;
        case 11:
                switch(i)
                {
                        case 0:
                                tecla='3';j=1;
                                break;
                        case 1:
                                tecla='6';j=1;
                                break;
                        case 2:
                                tecla='9';j=1;
                                break;
                        case 3:
                                tecla='l';j=1; //limpiar
                                break;
                        default:
                                break;
                }
                break;
        case 7:
                switch(i)
                {
                        case 0:
                                tecla='s';j=1; //seno
                                break;
                        case 1:
                                tecla='c';j=1; //coseno
                                break;
                        case 2:
                                tecla='t';j=1; // tangente
                                break;
```

```
                                                case 3:
                                                        tecla='e';j=1; //enter
                                                        break;
                                                default:
                                                        break;
                                                }
                                                break;
                                        default:
                                                break;
                                        }
                                usleep(150000);
                        }
                        i=i+1;
                        if(i==4){i=0;}
                        if(j==1){break;}
                }
        return tecla;
}

void lcd_clear_screen(alt_up_character_lcd_dev * lcd,int len, int delay1, int delay2){
        usleep(delay1*100000);
        alt_up_character_lcd_set_cursor_pos(lcd, 0, 0);
        alt_up_character_lcd_write(lcd, "                ", len);
        alt_up_character_lcd_set_cursor_pos (lcd, 0, 1);
        alt_up_character_lcd_write(lcd, "                ", len);
        usleep(delay2*100000);
}

char * convertIntChar(int num){
        char numConv[5]="",temporal;
        int car,i=0,longitud,j;
        char valCon;
         while(num!=0){
                car = num % 10;
                valCon = intToChar(car);
                numConv[i]= valCon;
                num = num / 10;
                i++;
         }
         longitud=strlen(numConv);
         for (i=0,j=longitud-1; i<longitud/2; i++,j--)
         {
                temporal=numConv[i];
                numConv[i]=numConv[j];
                numConv[j]=temporal;
         }
 return numConv;
}
```

```
int esdigito(char tec){
	if(tec=='1' || tec=='2' || tec=='3' || tec=='4' || tec=='5' || tec=='6' || tec=='7' || tec=='8' ||
tec=='9' || tec=='0')
		return 1;
	else
		return 0;
}

int char2int(char c){
	if(c=='1'){return 1;}
	else if(c=='2'){return 2;}
	else if(c=='3'){return 3;}
	else if(c=='4'){return 4;}
	else if(c=='5'){return 5;}
	else if(c=='6'){return 6;}
	else if(c=='7'){return 7;}
	else if(c=='8'){return 8;}
	else if(c=='9'){return 9;}
	else if(c=='0'){return 0;}
	else {return -1;}
}

int string2int( char c[]){
	int i=0,total=0,x=0;
	for(i=0;i<5;i++){
		x=char2int(c[i]);
		if(x>=0)
			total=(total*10)+x;
	}
	return total;
}

char* dispTop(char fun,char signo,char numero[5],int digi){
	char *linea="";
	int i=0,blancos=0;
	char blan[7]="      ";
		if(fun=='s'){
			snprintf(linea,5,"Sen(");
		}
		else if(fun=='c'){
			snprintf(linea,5,"Cos(");
		}
		else if(fun=='t'){
			snprintf(linea,5,"Tag(");
		}
		if(signo=='a'){
			strcat(linea,"+");
		}
		else if(signo=='r'){
```

```
                        strcat(linea,"-");
                }
        strncat(linea,numero,digi);
        blancos=15-7-digi;
        strcat(linea,")=");
        strncat(linea,blan,blancos);
        return *linea;
}

char* dispBot(float res,char signo,int cuadrante){
        char *linea="";
        if(signo=='+'){
                snprintf(linea,8,"+%1.6f",res);
        }
        else if(signo=='-'){
                snprintf(linea,8,"-%1.6f",res);
        }
        strcat(linea,"       ");
        return *linea;
}

int to_angulo_menor90(int angulo)
{
        do
        {
          if(angulo>=360)
                angulo=angulo-360;
          else
          {
                if(angulo>=270)
                        angulo=360-angulo;
                else
                {
                        if(angulo>=180)
                                angulo=angulo-180;
                        else
                        {
                                if(angulo>90)
                                        angulo=180-angulo;
                        }
                }
          }
        }while(angulo>90);
        return angulo;
}

int Get_cuadrante(int angulo)
{
        int cuadrante=0;
```

```
	do
	{
		if(angulo>=360)
			angulo=angulo-360;
		else
		{
			if(angulo>=270)
			{
				angulo=angulo-270;
				cuadrante=4;
			}
			else
			{
				if(angulo>=180)
				{
					angulo=angulo-180;
					cuadrante=3;
				}
				else
				{
					if(angulo>90)
					{
						angulo=angulo-90;
						cuadrante=2;
					}
				}
			}
		}
	}while(angulo>90);
	if(cuadrante==0)
	{
		cuadrante=1;
	}
	return cuadrante;
}


int binario_to_decimal(char resultado_binario[]){
	int i, j=-1;
	int resultado_decimal=0;
	for(i=15;i>=0;i--)
	{
		j++;
		if(resultado_binario[i]=='1')
			resultado_decimal=(int) (resultado_decimal+pow(2,j));
	}
	return resultado_decimal;
}
```

```
float resultado_funcion(int num_bin)
{
	int n;
	float result;
	n=(int)(pow(2,15));
	result=(float)(num_bin)/n;
	return result;
}

char get_signo(int cuadro, char tecla){
	char signo='+';
	if(cuadro==1)//los resultados positivos
	{}
	if(cuadro==2 && tecla!='s')//seno es negativo
		signo='-';
	if(cuadro==3 && tecla!='t')//tan en negativo
		signo='-';
	if(cuadro==4 && tecla!='c')//cos en negativo
		signo='-';
	return signo;
}

int Get_cuadrante2(int angulo, char signo_in)
{
  int cuadrante=0;
	do
	{
		if(angulo>=360)
		{
			angulo=angulo-360;
		}
		else
		{
			if(angulo>=270)
			{
				angulo=angulo-270;
				if(signo_in=='+'){cuadrante=4;}
				else{cuadrante=1;}
			}
			else
			{
				if(angulo>=180)
				{
					angulo=angulo-180;
					if(signo_in=='+'){cuadrante=3;}
					else{cuadrante=2;}
				}
				else
				{
```

```
                                        if(angulo>90)
                                        {
                                                angulo=angulo-90;
                                                if(signo_in=='+'){cuadrante=2;}
                                                else{cuadrante=3;}
                                        }
                                }
                        }
                }
        }while(angulo>90);
        if(cuadrante==0)
        {
                if(signo_in=='a'){cuadrante=1;}
                                                else{cuadrante=4;}
        }
  return cuadrante;
}
```

BIBLIOGRAFÍA

[1] Enciclopedia libre, Sistemas en chip.

http://es.wikipedia.org/wiki/System_on_a_chip

Fecha de consulta: Agosto 2012

[2] David A. Pérez A., Sistemas Embebidos y Sistemas Operativos Embebidos.

www.ciens.ucv.ve/escueladecomputacion/documentos/archivo/88

Fecha de consulta: Septiembre 2012

[3] Enciclopedia libre, FPGA.

http://es.wikipedia.org/wiki/Field_Programmable_Gate_Array

Fecha de consulta: Noviembre 2012

[4] Jorge Rodríguez Araújo, Microprocesador NIOS II.

http://es.scribd.com/doc/28358833/Estudio-del-microprocesador-Nios-II

Fecha de consulta: Octubre 2012

[5] Altera, Procesador Nios II.
http://www.altera.com/devices/processor/nios2/ni2-index.html
Fecha de consulta: Septiembre 2012

[6] Enciclopedia libre, Quartus II.
http://es.wikipedia.org/wiki/Quartus_II
Fecha de consulta: Enero 2013

[7] Tanya Vladimirova and Hans Tiggeler, Implementación en FPGA de generadores de seno y coseno usando el algoritmo CORDIC.
http://www3.matapp.unimib.it/corsi-2005-2006/files/vladimirova-tiggler--fpga-implementation.pdf
Fecha de consulta: Enero 2013

[8] Richard Herveille. Algoritmo CORDIC.
http://cutler.eecs.berkeley.edu/classes/ee225c/Papers/cordic.pdf
Fecha de consulta: Enero 2013