ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"DISEÑO DE PRÁCTICAS PARA EL APRENDIZAJE DE SISTEMAS EMBEBIDOS BASADOS EN EL PROCESADOR NIOS® II UTILIZANDO HERRAMIENTAS DE QUARTUS II Y LA TARJETA DE0-NANO"

INFORME DE PROYECTO INTEGRADOR

Previo a la obtención del Título de:

INGENIERO EN ELECTRICIDAD ESPECIALIZACIÓN ELECTRÓNICA Y AUTOMATIZACIÓN INDUSTRIAL

PRESENTADO POR:

VÍCTOR ALBERTO ARCENTALES DUEÑAS

Guayaquil – Ecuador

AÑO: 2016

AGRADECIMIENTO

Mis más sinceros agradecimientos a Dios y mi familia por ser un eje fundamental y constante en mi desarrollo integral. Además agradezco a los profesores que han sido parte de mi formación académica, desde mi escuela y colegio por 13 años “Domingo Comín”, hasta mi nivel académico actual en ESPOL por 5 años. Un extensivo agradecimiento a aquellos compañeros y amigos con los que hemos compartido durante nuestro crecimiento académico hasta la fecha.

Víctor

DEDICATORIA

El presente proyecto lo dedico a mis padres y hermanos por el acompañamiento diario durante mi vida. También me gustaría dedicarles a todas aquellas personas que han contribuido en el desarrollo y culminación de este trabajo, en especial a Mishelle que aportó activamente en el mismo.

Víctor Arcentales Dueñas

TRIBUNAL DE EVALUACIÓN

Wilton Agila, Ph.D.

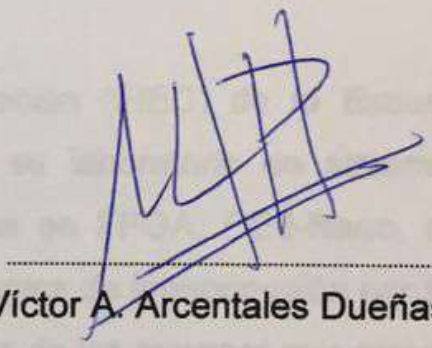
PROFESOR EVALUADOR

Ronald Ponguillo, MSc.

PROFESOR EVALUADOR

DECLARACIÓN EXPRESA

"La responsabilidad y la autoría del contenido de este Trabajo de Titulación, me corresponde exclusivamente; y doy mi consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"



Víctor A. Arcentales Dueñas

RESUMEN

Los sistemas embebidos son un campo que gana acogida de forma acelerada y dentro de esta área las FPGAs (Field Programmable Gate Arrays) han alcanzado gran participación gracias a su flexibilidad, debido a que permite definir su función luego de ser fabricado y más importante aún, da la posibilidad de corrección del diseño en caso de ser requerido luego de evaluar su funcionamiento en el campo sin incurrir en elevados costes.

La Facultad de Ingeniería en Electricidad y Computación (FIEC) de la Escuela Superior Politécnica del Litoral (ESPOL), cuenta en su laboratorio de sistemas digitales con tarjetas de desarrollo educativas basadas en FPGA, DE0-Nano, sin embargo las mismas son empleadas únicamente para lógica de interconexión por los estudiantes de la materia práctica y se aprovechan pocos de los recursos que provee esta tarjeta.

El presente proyecto pretende ofrecer una guía para el aprendizaje de sistemas embebidos basándose en el procesador Nios II con el que cuenta la FPGA de la DE0-Nano mediante el uso de los chips integrados en esta tarjeta de desarrollo y más aún, establecer una base sobre la cual los estudiantes de pregrado puedan elaborar proyectos en esta temática de alcances similares a los que se lleva acabo en universidades de todo el mundo.

La metodología de aprendizaje de la presente propuesta esta basada en el desarrollo de seis prácticas, mediante las cuales se adquieren los conocimientos de los principios básicos del desarrollo de sistemas embebidos sobre el procesador Nios II, se aprende el acceso a registros de memoria que se les asigna a los componentes en la etapa de creación de hardware y al uso de periféricos propios de la tarjeta o externos.

El capítulo 1 presenta la descripción del problema, los objetivos generales y específicos, en detalle la metodología implementada y los alcances y limitaciones del presente proyecto.

El capítulo 2 describe las herramientas de software que se utilzan durante el desarrollo de las prácticas, el flujo de diseño que se lleva a cabo en las mismas, detallando la creación de hardware en Qsys y las aplicaciones de software en Nios II SBT para eclipse y también se presenta información pertinente a los periféricos que forman parte de las prácticas.

El capítulo 3 describe la composición y el aporte de cada práctica al flujo de aprendizaje de sistemas embebidos y brevemente presenta un análisis de los costos de implementación de las prácticas propuestas.

En la parte de anexos se presenta el material completo de las prácticas, tanto el archivo que contiene en detalle el desarrollo de las mismas y los archivos que presentan la información teórica que sirve de fundamentación para un mejor entendimiento y aprovechamiento de las prácticas y por lo cual se lo plantea como requisitos previos a relizarlas.

ÍNDICE GENERAL

CAPÍTULO 1

1. GENERALIDADES.

1.1 Descripción del Problema

La industria moderna cada vez presenta más variedades de entornos demandantes, a los que países en vía de desarrollo deben apuntar, dejando a un lado soluciones tecnológicamente desactualizadas con las que por ejemplo, la industria ecuatoriana cuenta en gran medida y formando personal calificado con amplios conocimientos en áreas tecnológicas de gran crecimiento a nivel mundial como lo es el desarrollo de sistemas embebidos.

A la fecha de realización de este proyecto la Facultad de Ingeniería en Electricidad y Computación (FIEC) de la Escuela Superior Politécnica del Litoral (ESPOL), no cuenta con un curso de sistemas embebidos. Aunque, se conoce que tiene entre sus próximos proyectos la implementación de un laboratorio sobre la temática mencionada, actualmente no se cuenta con estudiantes capacitados en esta línea de tal manera que puedan explotar los beneficios que dicho ambiente de trabajo les proveería y mucho menos que puedan dar soluciones industriales con este tipo de sistemas en diversas áreas como comunicaciones y sistemas de control específicos.

1.2 Justificación

Los sistemas embebidos son una disciplina que está involucrada en innumerables áreas y es un tema fundamental del que todo ingeniero electrónico debe conocer y poder hacer uso para generar prototipos o implementar aplicaciones en la industria.

Los estudiantes de pregrado, especialmente en las carreras de electrónica requieren de una guía para incursionar en el desarrollo de hardware y creación de software, tomando como punto de partida la condición actual y particular en cuanto a los conocimientos provistos en las diferentes materias que se ven, sobre todo en los cursos de sistemas digitales.

1.3 Objetivos

1.3.1 Objetivo General

- Diseñar e implementar prácticas con *Qsys*, *Nios II Software Build Tools for Eclipse* y la tarjeta DE0-Nano, para el aprendizaje de sistemas embebidos basados en el procesador Nios® II.

1.3.2 Objetivos Específicos

Los objetivos específicos están basados en que las prácticas a desarrollar permitan adquirir los siguientes conocimientos a su lector:

- Entender y llevar a cabo el flujo adecuado para el diseño de hardware y software en un sistema de Nios® II.
- Configurar la FPGA con un diseño de hardware que contenga un procesador Nios® II, utilizando Qsys.
- Aprovechar la flexibilidad de modificar el diseño fácilmente para personalizar y acelerar el uso de la FPGA.
- Emplear en las prácticas la mayoría de los recursos provistos en la tarjeta de desarrollo DE0-Nano.

1.4 Metodología

Para el desarrollo de las prácticas se tomó en consideración los conocimientos previos que se adquieren en las materias teóricas y experimentales dentro del área de sistemas digitales, en la FIEC, y estos marcan el punto de partida; es decir, se supone que el lector posee una base sobre la cual agregar el aprendizaje que este proyecto pretende. Por lo antes expuesto se emplea la tarjeta DE0-Nano y el programa de diseño para dispositivos programables producido por Altera, Quartus II, en su versión Web Edition, la misma que es gratuita y permite trabajar con limitaciones que no afectan el alcance del presente proyecto.

Las prácticas son debidamente documentadas, indicando los objetivos de cada una, las prácticas previas que se deben haber realizado y los documentos de referencia que sirven de soporte para el desarrollo de la misma.

1.5 Alcances y Limitaciones del Proyecto

Alcances:

- El presente trabajo es una base para el desarrollo de sistemas embebidos, con prácticas desarrolladas únicamente en la tarjeta DE0-Nano de Altera.
- El flujo de desarrollo de los sistemas embebidos que se presenta en este trabajo se enfoca en el procesador Nios II de Altera, sin embargo gran parte de los conceptos son aplicables a otros fabricantes.

Limitaciones:

- La capacidad de memoria, elementos lógicos y periféricos contenidos en la tarjeta DE0-Nano.

CAPÍTULO 2

2. DESCRIPCIÓN DE HERRAMIENTAS DE QUARTUS II Y LA TARJETA DE0-NANO PARA EL DESARROLLO DE APLICACIONES EMBEBIDAS.

2.1 INTRODUCCIÓN A SISTEMAS EMBEBIDOS

Los sistemas embebidos pueden ser descritos como sistemas computacionales dedicados y dirigidos a aplicaciones específicas, que se diferencian de otros tipos de sistemas computacionales debido a las restricciones que deben cumplir, de manera general con respecto a calidad, fiabilidad, seguridad, tamaño, peso, etc., específicamente limitaciones de hardware como el rendimiento de procesamiento, consumo de energía, memoria, etc., y en cuanto a software como limitaciones o inexistencia de sistema operativo y aplicaciones a escala reducida.

Los sistemas embebidos contienen al menos un componente programable y requieren continua interacción con el entorno en tiempo real [1], [2]. Dentro de los sistemas embebidos se encuentran las FPGA que es una tecnología relativamente joven y que se mantiene en constantes mejoras, permitiéndole alcanzar más aplicaciones con el transcurso del tiempo.

2.1.1 FPGA (Field Programmable Gate Array)

Es un circuito integrado programable cuyos recursos internos consisten de una matriz de bloques lógicos configurables (CLBs) rodeados por bloques de entradas y salidas (I/O) periféricas y cuya interconexión puede ser programada, en la Figura 2.1, se presenta la integración de los recursos internos de las FPGAs. La diferencia de las FPGAs con los circuitos integrados de aplicación específica (ASICs), los cuales son construidos para un diseño particular, es que en las primeras se puede cambiar su diseño incluso después de tener el producto terminado e implementado en el campo [3].

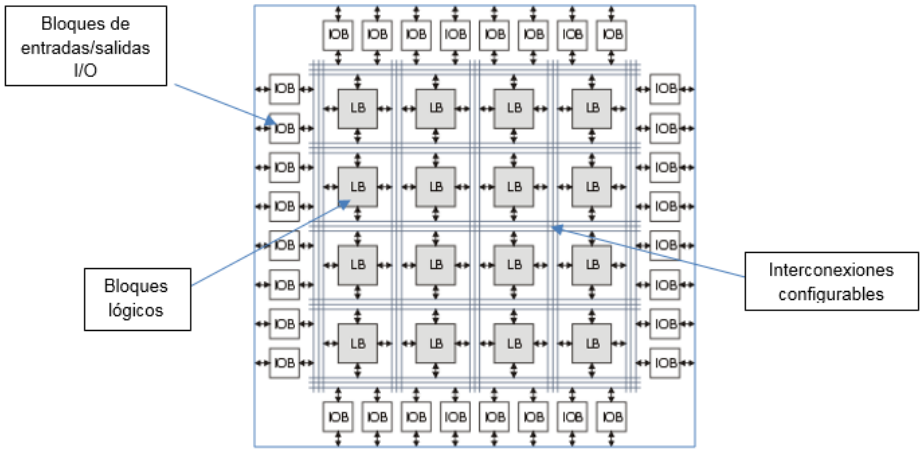


Figura 2.1 Recursos internos de las FPGAs

2.1.2 Ventajas del desarrollo de sistemas embebidos basados en FPGA con respecto a otras alternativas.

Existen en el mercado tarjetas de desarrollo con gran número de adquisiciones y que ha generado la creación de grupos para la socialización de proyectos personales realizados sobre dichas tarjetas, tales como Arduino y Raspberry PI.

Es muy probable que con estas tarjetas se pueda llevar a cabo tareas similares que las desarrolladas con tarjetas basadas en FPGA, sin embargo su principal diferencia radica en la arquitectura de los procesadores que emplean. Mientras que estas tarjetas presentan una arquitectura definida que incluso se puede encontrar en las hojas de especificaciones de los mismos, ya sea los microcontroladores de ATMega328 de Arduino UNO o el procesador ARM1176JZF-S de Raspberry PI, los sistemas que cuentan con procesadores como Nios II y otras versiones del ARM que soportan familias de las FPGAs, son mucho más flexibles al poseer una arquitectura configurable que se puede desarrollar en base a los requerimientos de ingeniería que se tengan.

Entre los aspectos más fuertes en el uso de FPGAs para el desarrollo de sistemas embebidos y que los vuelve bastante empleados en aplicaciones donde es necesario un alto desempeño tenemos [4]:

- **Paralelismo.-** Al utilizar HDL (Hardware Description Language) para implementar la lógica apropiada, es diferente que las listas secuenciales en lenguaje C o ensamblador. Esto significa que múltiple ocupaciones ocurren en paralelo.

- **Alto desempeño y fiabilidad.-** Debido a que no se cuenta con sistema operativo sobre el chip FPGA, el código es implementado con el propósito de asegurar el máximo desempeño y fiabilidad.

- **Alto Determinismo.-** Se ejecutan algoritmos a velocidades que pueden ser determinadas de hasta 20ns (o más rápido en algunos casos dependiendo del reloj empleado).

- **Reconfigurable.-** Permite adecuar las conexiones de hardware e incorporación de chips externos o alterar sistemas ya existentes en base a los requerimientos que se tienen.

2.1.3 Tarjeta de desarrollo DE0-Nano

La tarjeta DE0-Nano introduce una plataforma de desarrollo de FPGA de tamaño compacto adecuado para una amplia gama de diseño de proyectos [5].

Las partes que componen la tarjeta DE0-Nano se presentan en la Tabla 2.1 y se observa la ubicación de cada una de las mismas en la Figura 2.2 y Figura 2.3, la vista superior y posterior de la tarjeta respectivamente.

CARACTERÍSTICA	DETALLES
FPGA	• Altera Cyclone® IV EP4CE22F17C6N • 153 pines de I/O de la FPGA
Elementos de configuración	• Circuito sobre la tarjeta USB-Blaster para programación • Spansion EPCS64
Cabeceras de expansión	• Dos cabeceras de 40 pines (GPIOs) • Una cabecera de 26 pines (8 entradas del ADC + 18 GPIOs)
Dispositivos de Memoria	• SDRAM de 32MB • EEPROM I2C 2Kb
Entradas y salidas (I/O) de uso general	• 8 LEDs verdes • 2 pulsadores con antirebote • 4 interruptores tipo DIP
Sensor giroscópico	• Acelerómetro de 3 ejes, ADI ADXL345, con alta resolución (13-bit)
Convertidor A/D	• NS ADC128S022, 8-Canales, convertidor de 12-bit A/D
Reloj del sistema	• Reloj oscilador de 50 MHz, sobre la tarjeta
Fuente de poder	• Puerto tipo mini USB de 5V • 2 pines de 5V por cada cabecera de expansión de entradas y salidas de propósito general. • 2 pines externos de alimentación (3.6-5.7V)

Tabla 2.1 Características de la tarjeta DE0-Nano

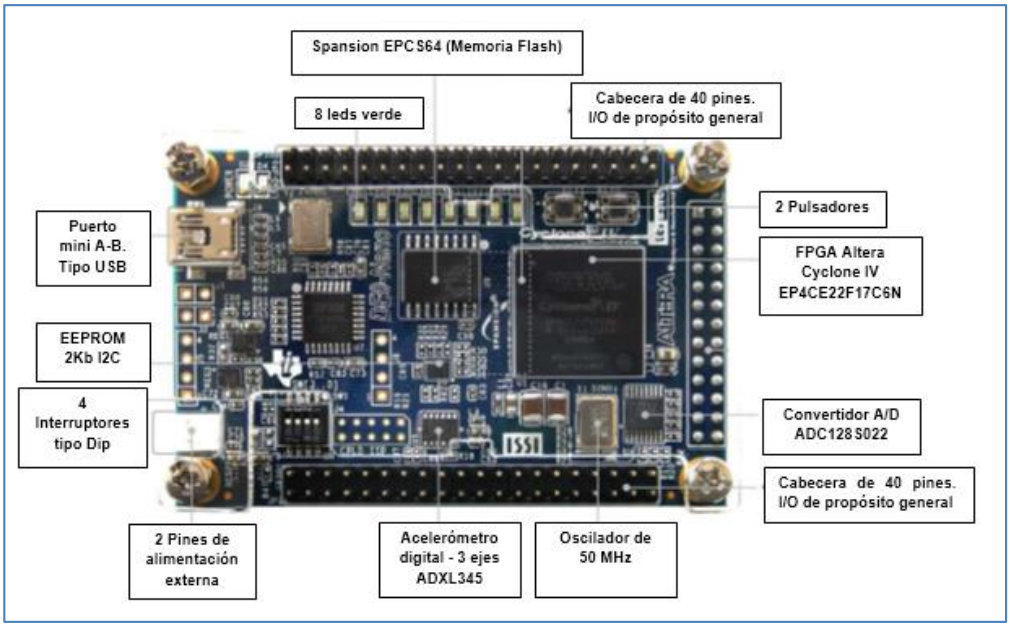


Figura 2.2 Vista superior de la tarjeta DE0-Nano con sus componentes [5].

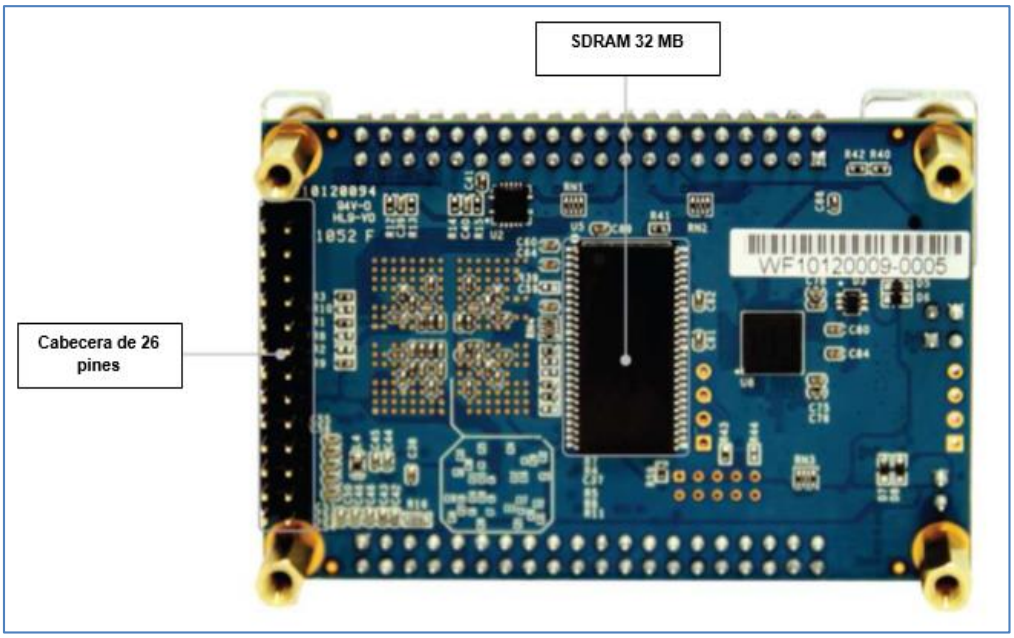


Figura 2.3 Vista posterior de la tarjeta DE0-Nano con sus componentes [5].

La tarjeta DE0-Nano provee máxima flexibilidad, todas las conexiones entre los componentes de la tarjeta son hechas mediante del dispositivo FPGA Cyclone IV como se observa en el diagrama de bloques de la Figura 2.4, Por lo tanto el usuario puede configurar la FPGA para implementar el diseño de cualquier sistema [5].

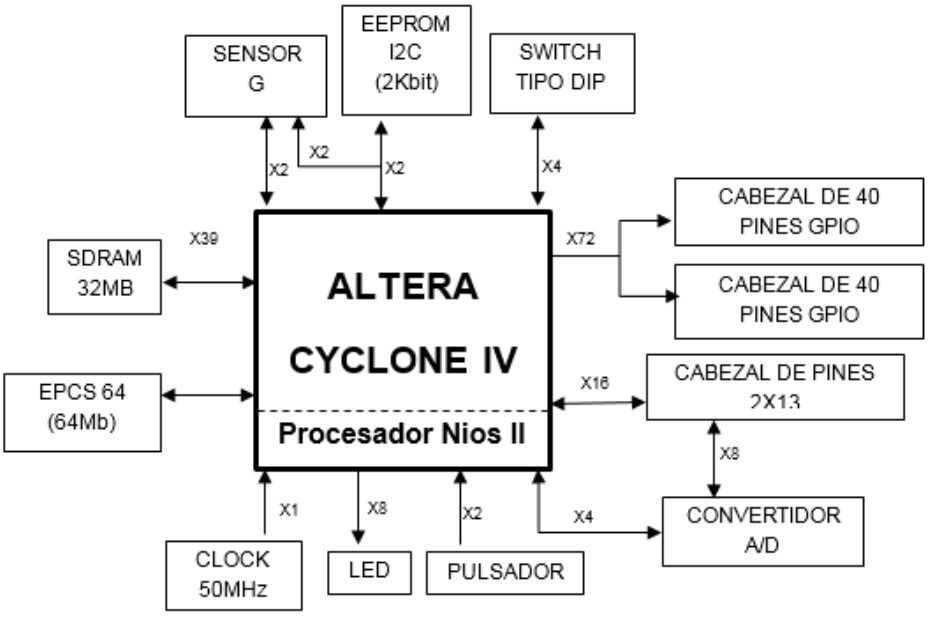


Figura 2.4 Diagrama de bloques de la tarjeta DE0-Nano [5].

Sobre todas las familias de FPGAs de Altera se puede instanciar el procesador Nios II que pertenece al mismo fabricante, por lo tanto esto incluye a la FPGA Cyclone® IV EP4CE22F17C6N de la tarjeta DE0-Nano y además permite visualizar que los sistemas a implementar pueden ser utilizados en otras tarjetas de desarrollo al menos de Altera sin mayores modificaciones.

2.1.4 Procesador Nios II

Es un procesador de propósito general diseñado con la arquitectura RISC (Reduced Instruction Set Computing), entre las ventajas que le provee esta arquitectura está la posibilidad de segmentación y paralelismo en la ejecución de instrucciones [6]. Este procesador puede ser instanciado en los dispositivos FPGA de Altera en conjunto con otros componentes periféricos dentro de un sistema completo como por ejemplo el que se visualiza en la Figura 2.5.

Entre algunas de las características con las que cuenta este procesador se tiene:

- Conjunto de instrucciones de 32 bits, ruta de datos y espacio de direcciones.
- 32 registros de propósito general.
- 32 fuentes de interrupción.
- Controlador de interrupciones externas para más fuentes de interrupciones.
- Opción de instrucciones de punto flotante para operaciones de punto flotante de precisión simple.
- Acceso a una variedad de periféricos on-chip, interfaces de memorias y periféricos off-chip.
- Desempeño por encima de los 250 DMIPS (Dhrystone Millions of Instructions Per Seconds)

Un sistema de procesador Nios II es equivalente a un microcontrolador que incluye un procesador, una combinación de periféricos y una memoria en un único chip. A diferencia de un microcontrolador es configurable y se puede añadir o remover características hasta alcanzar el rendimiento o

metas de costo. El núcleo del procesador no está fijo en silicio y puede ser incorporado en cualquier tarjeta de la familia Altera [6].

2.1.5 Estructura de interconexión del sistema

La estructura de interconexión del sistema (*System Interconnect Fabric*) es la colección de interconexiones y recursos lógicos que conectan las interfaces de los componentes en un sistema como se visualiza en el ejemplo de la Figura 2.5. En la herramienta Qsys que se describe en la sección 2.2 se genera la estructura de interconexión del sistema para satisfacer las necesidades de los componentes del mismo. Esta interconexión garantiza que las señales están conectadas correctamente entre el maestro y los esclavos.

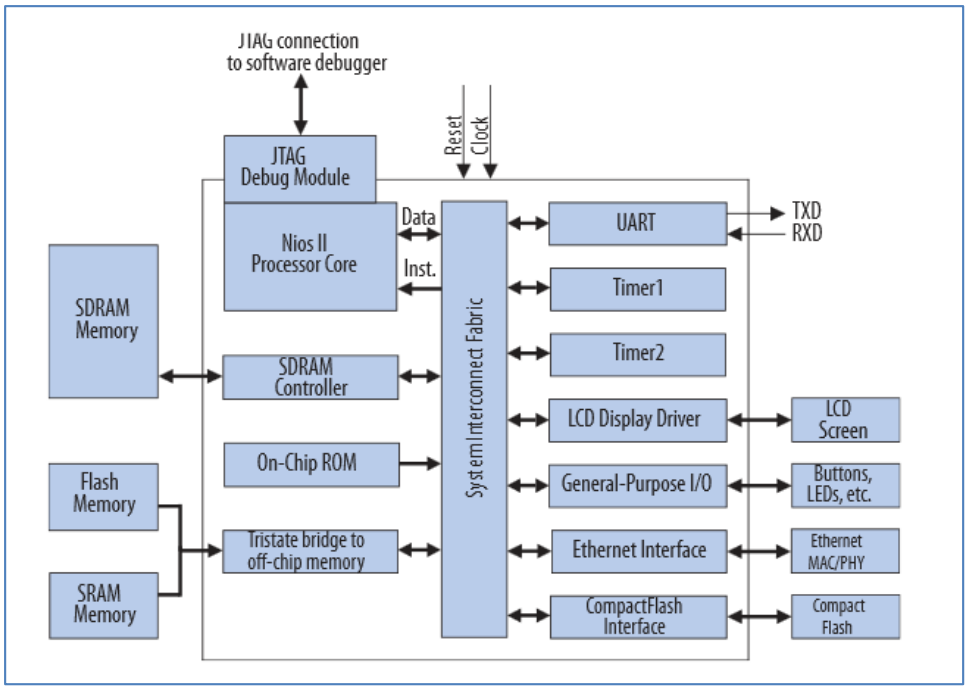


Figura 2.5 Ejemplo de un sistema de procesador Nios II [6].

2.2 IMPLEMENTACIÓN DE HARDWARE: QSYS

Qsys es una herramienta que se encuentra dentro del programa de Quartus II, y que se utiliza para diseñar el sistema de hardware digital, en la *Figura 2.6* se observa la ventana que aparece al abrir Qsys, en esta herramienta se seleccionan los componentes que se desean en el sistema, tales como procesadores, memorias, interfaces de entrada y salida, temporizadores, etc., en una interfaz gráfica y amigable al usuario. Al final del diseño se genera el lenguaje de descripción de hardware (HDL) que conecta todos los componentes [8].

Figura 2.6 Pantalla al abrir la herramienta Qsys

2.2.1 Interfaces Avalon

Las interfaces Avalon simplifican el diseño del sistema permitiendo conectar fácilmente componentes en una FPGA de Altera. Estas interfaces están diseñadas dentro de los componentes disponibles en Qsys. También se pueden usar estas interfaces estándares en componentes personalizados [7].

Las interfaces Avalon disponibles son:

- **Avalon Streaming Interface (Avalon-ST). -** Es una interfaz que soporta el flujo de datos unidireccionales, incluyendo flujos de datos multiplexados, paquetes y datos DSP (Digital Signal Processing).

- **Avalon Memory Mapped Interface (Avalon-MM). -** Típica interfaz basada en direcciones de lectura y escritura de conexiones maestro-esclavo.

- **Avalon Conduit Interface.-** Tipo de interfaz que acomoda señales individuales o grupos de señales que no encajan dentro de otros tipos Avalon.

- **Avalon Tri-State Conduit Interface (Avalon TC). -** Interfaz que da soporte a conexiones de periféricos off-chip, permitiendo así que compartan pines a través de la multiplexación de señales, reduciendo el uso de pines de la FPGA.

- **Avalon Interrupt Interface.** – Una interfaz que permite a los componentes señalar eventos en otros.

- **Avalon Clock Interface.** – Una interfaz que dirige o recibe relojes.

- **Avalon Reset Interface.** – Una interfaz que provee conectividad con el reset.

2.2.2 Intellectual Property (IP) Cores

Un bloque o núcleo de propiedad intelectual es un subcircuito prediseñado para utilizarlo en grandes diseños. Altera provee IP cores que tienen soporte en los diversos dispositivos de las tarjetas FPGA de este fabricante.

Los IP cores están optimizados para los dispositivos de Altera y pueden ser fácilmente implementados para reducir el tiempo de diseño y prueba.

Utilizando el editor de parámetros de Qsys se puede añadir IP cores al sistema, configurarlos y especificar su conectividad [9].

2.2.3 Descripción breve de los componentes utilizados propios de Qsys

Se describe a continuación los componentes utilizados en las prácticas que por defecto se encuentran en las librerías de Qsys y que no requieren de alguna instalación adicional:

- **Nios II Core.-** Es un módulo de un '*soft-processor*'; es decir un procesador que puede ser enteramente implementado utilizando un proceso que convierte un circuito de forma abstracta a un diseño en términos de puertas lógicas.

- **On-Chip Memory.-** Almacena datos de manera temporal y provee un control del flujo en un sistema Qsys. Sistemas de procesadores requieren al menos una memoria para los datos y las instrucciones. La DE0-Nano tiene 594 Kilobits (Kb) de esta memoria embebida en la FPGA, equivalente a 74.25KiloBytes (KB).

- **JTAG UART.-** Implementa un método para comunicar de forma serial el flujo de caracteres entre la PC y un sistema de Qsys sobre una FPGA de Altera. En muchos casos elimina la necesidad adicional de tener una conexión serial RS-232. Los periféricos maestros como el procesador Nios II se comunica con el JTAG UART leyendo y escribiendo registros de control y datos. La PC puede conectarse a la FPGA mediante un cable JTAG de Altera, como el USB-Blaster (Se conoce con este nombre a la circuitería que provee un enlace USB y a su software asociado).

- **Interval Timer**.- La mayoría de los sistemas de control requieren de un temporizador para precisar el cálculo del tiempo. Este bloque provee de contadores de 32 y 64 bits, control de inicio, parada y reset del temporizador, característica opcional de perro guardián

que resetea el sistema si el temporizador alcanza cero. Es compatible con procesadores de 32 y 64 bits.

- **System ID Peripheral.**- Este bloque protege contra accidentalmente descargar software compilado para un sistema de Nios II diferente, para llevar a cabo esta tarea le provee al sistema Qsys con un identificador único.

- **PIO (Parallel Input/Output).-** Provee un método sencillo para que el procesador del Sistema pueda recibir señales de entradas y manejar las salidas. Permite una interfaz entre el puerto esclavo de la memoria mapeada Avalon y los puertos de entrada/salida de propósito general. Los puertos I/O pueden ser conectados tanto a la memoria (on-chip) o los pines I/O que conectan dispositivos externos con la FPGA.

- **SDRAM Controller.-** Provee una interfaz Avalon – MM a la SDRAM off-chip con la que cuenta la tarjeta DE0-Nano. Este componente permite a los diseñadores crean sistemas en las tarjetas de Altera que se conecten fácilmente con los chips SDRAM.

2.2.4 Descripción breve de los componentes utilizados de University Program

Para acceder a componentes adicionales a los que por defecto se encuentran en Qsys debe instalarse el University Program que contiene los bloques de propiedad intelectual que dan soporte a los diversos dispositivos de las tarjetas basadas en FPGA de Altera, algunos de los cuales se utilizan en las prácticas del presente proyecto, los mismos se detallan a continuación:

- **DE0-Nano ADC Controller.-** Provee una interfaz entre el procesador Nios II y el chip convertidor analógico-digital (NS ADC128S022) que se encuentra en la tarjeta DE0-Nano. Mediante

este componente se puede indicar cuantos canales (entre los 8 disponibles) se van a utilizar. El ADC lee los valores analógicos y los convierte en valores digitales de 12 bits.

- **Clock Signals. -** Provee señales de reloj para componentes que tienen requerimientos más específicos en cuanto a la frecuencia como por ejemplo el SDRAM Controller de la DE0-Nano, que requiere para una operación apropiada adelantar al reloj del sistema Nios II por 3 nanosegundos.

- **Parallel Port. -** Provee una interfaz simple para entradas y salidas de propósito general. Es una versión del núcleo PIO previamente estudiado; pero adaptado al uso específico de las tarjetas de la serie DE en base a las entradas y salidas que tienen integradas.

- **Accelerometer SPI Mode Core.-** Provee una interfaz conveniente para acceder al chip del acelerómetro, ADXL345, el mismo que mide aceleración en tres ejes (x, y, z) y que en estado estacionario la aceleración medida es debida a la gravedad, los resultados de las medidas se muestran en 13 bits de resolución en un rango de ±16g. Este chip está integrado en la tarjeta DE0-Nano y la tarjeta VEEK-MT. El chip puede ser accesado usando la comunicación SPI (Serial Peripherial Interface Bus) en un arreglo de tres hilos o una interfaz I2C de dos hilos.

2.2.5 Tipos de archivos generados en Qsys más importantes

En la Tabla 2.2 se describen los archivos o directorios más importantes que genera Qsys. Aunque todos los archivos generados constituyen el todo de nuestro sistema de hardware, los archivos que se muestran a continuación son aquellos a los cuales se hacen mención en etapas posteriores para el entendimiento o uso en el desarrollo del sistema embebido basado en el procesador Nios II en su totalidad [10].

Nombre del archivo o del directorio	Descripción
<diseño_qsys>	Es el directorio que contiene todo el proyecto.
<diseño_qsys>**.bsf**	Es la representación del sistema de Qsys en forma de bloque para utilizarlo en Quartus II Block Diagram Files (.bdf)
<diseño_qsys>**.html**	Es un reporte del sistema que contiene información sobre las conexiones externas del sistema, las direcciones de cada componente respecto a las direcciones del maestro y los parámetros asignados para cada componente.
<diseño_qsys>**.sopcinfo**	Describe los componentes y conexiones en el sistema. Este archivo es una completa descripción del sistema.
/synthesis	Este directorio contiene todos los archivos de salida generados por Qsys usados al sintetizar el diseño.
<diseño_qsys>**.v**	Contiene un archivo en HDL del más alto nivel del sistema que instancia cada componente en el sistema.
<diseño_qsys>**.qip**	Este archivo contiene el software de quartus II necesitado para compilar el diseño. Este archivo debe ser añadido al proyecto.
/submodules	Contiene los códigos verilog HDL o VHDL de cada submódulo para la síntesis.

Tabla 2.2 Archivos y directorios generados en Qsys

2.3 IMPLEMENTACIÓN DE SOFTWARE

Sobre el sistema de hardware diseñado en Qsys, se procede a correr una aplicación de software que hace uso de los componentes que contiene dicho sistema. Para la creación del software Altera provee de la herramienta Nios II Software Build Tools (SBT), mediante dos posibles interfaces: Nios II SBT Command Line y Nios II SBT for Eclipse. Debido a que la primera interfaz requiere del conocimiento de comandos específicos, para el desarrollo de las prácticas se emplea la segunda opción, la misma que se describe a continuación:

Nios II Software Build Tools (SBT) for Eclipse

Es un conjunto de módulos basados en la plataforma de Eclipse y los módulos de las herramientas de desarrollo de Eclipse C/C++ (CDT). El Nios II SBT provee una plataforma consistente que trabaja para todos los sistemas de procesador embebidos Nios II. Se puede llevar a cabo en este entorno todas las tareas de desarrollo de software tales como creación, edición, construcción, ejecución, depuración y mediciones dinámicas de programas [11].

Es importante entender que Eclipse es un programa compuesto de un conjunto de herramientas de programación de código abierto que suele ser usado para desarrollar entornos de desarrollo integrado [12], y justamente esta es la utilidad que le dio Altera para tener su propio entorno de desarrollo de software para sus procesadores Nios II. En la Figura 2.7 se observa la ventana que aparece al abrir la herramienta Nios II SBT for Eclipse.

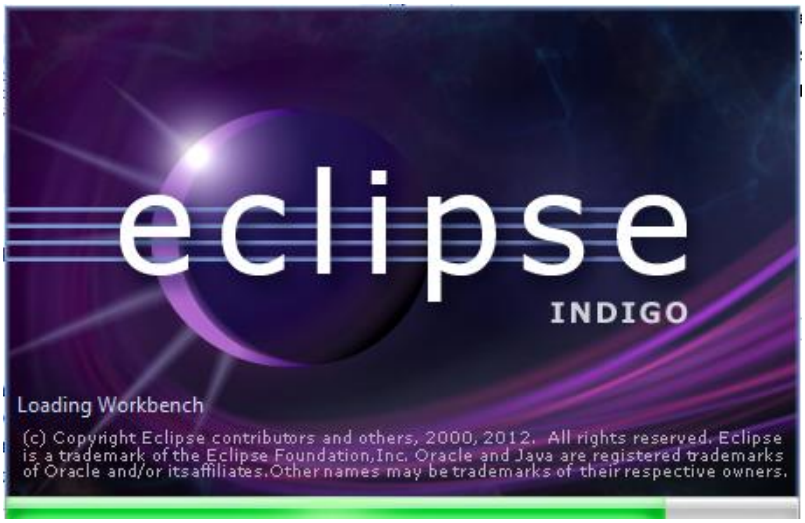


Figura 2.7 Pantalla al abrir Nios II Software Build Tools for Eclipse

Nios II SBT permite que el área de trabajo contenga las ventanas necesarias en base al tipo de tarea que se realiza. Es para ello que cuenta con la selección de *perspectivas,* antes de crear un proyecto de Nios II debemos asegurarnos que la perspectiva Nios II este visible.

Tipos de proyectos de software Nios II

Cada programa Nios II que se desarrolla consiste de un proyecto de aplicación, un proyecto de librería de usuario y un proyecto BSP. Al construir el programa Nios II se crea un archivo ejecutable de extensión .elf (Executable and Linking Format File) el mismo que corre sobre el procesador Nios II [11].

- **Proyecto de Aplicación**

  Este proyecto consiste en una colección de código fuente y un *makefile*, que es el componente principal del proyecto de software Nios II, este archivo describe todos los componentes de software del proyecto y como son compilados y enlazados.

- **Proyecto de librería de usuario**

Es una colección de código fuente compilado para crear un único archivo de librería (**.a**). A menudo este tipo de proyecto contiene funciones de propósito general y reusable como por ejemplo funciones aritméticas comunes.

- **Proyecto BSP**

Es una librería especializada que contiene código de soporte específico del sistema. Un BSP contiene los siguientes elementos: librería de funciones primitivas conocidas como HAL (Hardware Abstraction Layer), librerías estándares de lenguaje C, controladores de los dispositivos, sistema operativo en tiempo real (opcional).

2.4 FLUJO DE DESARROLLO PARA LA CREACIÓN DE UN SISTEMA NIOS II

El desarrollo de un sistema basado en el procesador Nios II consiste de dos etapas fundamentales que son: el diseño del hardware y la creación del software. Una vez que se tienen ambas se puede hacer una integración entre ellas para tener diseñado el sistema completo. El flujo del desarrollo del sistema se presenta en la Figura 2.8 y se resume en los siguientes pasos [13]:

1. Se inicia el diseño *analizando los requerimientos del sistema*; es decir las necesidades generales, como que procesador utilizar (/e, /s, /f), que componentes requiere el sistema y cuantos, entre otros.

2. Una vez que se conocen los requerimientos, *se define y genera el sistema en Qsys*, con esta herramienta se especifica el núcleo del procesador Nios II, la memoria y otros componentes que se necesitan.

3. Se debe *integrar el sistema dentro del proyecto de Quartus II* porque usando este programa se llevan a cabo las tareas para crear el diseño de hardware de FPGA final, además puede añadirse lógica externa al sistema Nios II.

En grandes proyectos se suelen dividir por equipos el desarrollo de hardware y la creación del software para correr sobre dicho hardware. Una vez que se tiene definido el sistema en Qsys, se puede *desarrollar el software con el Nios II Software Build Tools for Eclipse* en paralelo a la integración del hardware en Quartus II.

4. El desarrollo se continúa dentro del entorno del programa Quartus II, donde se realiza la *asignación de los pines, requerimientos de tiempo y otras restricciones*; es decir, se identifica y destina los pines a usar de la tarjeta de desarrollo, se considera las necesidades específicas de ciertos componentes en cuanto a tiempo y alguna otra condición para su correcto funcionamiento.

5. Una vez culminados todos los ajustes de hardware se procede *a compilar el diseño de hardware para la tarjeta de desarrollo* que se está utilizando y corregir cualquier clase de errores que se pudiera generar durante este proceso.

6. Luego que la compilación sea exitosa se *descarga el diseño de la FPGA a la tarjeta de desarrollo*, para ello se hace uso de la herramienta de programación del programa Quartus II y los archivos generados luego de la compilación.

7. Se *descarga el ejecutable del software (.elf) al sistema Nios II cargado en la tarjeta*, este archivo se obtiene al construir el software en Nios II SBT para Eclipse.

8. Una vez realizada correctamente la construcción del software se *corre y depura sobre la tarjeta de desarrollo* para observar y evaluar la correcta operación del sistema en base a lo planteado inicialmente.

9. Una vez corrido el programa sobre la tarjeta, en caso de encontrar alguna posible mejora para el desempeño del sistema se puede regresar a los pasos de diseño de software o hardware y realizar dichas mejoras, de esta manera se *refina el software y hardware en base a los requerimientos propuestos*.

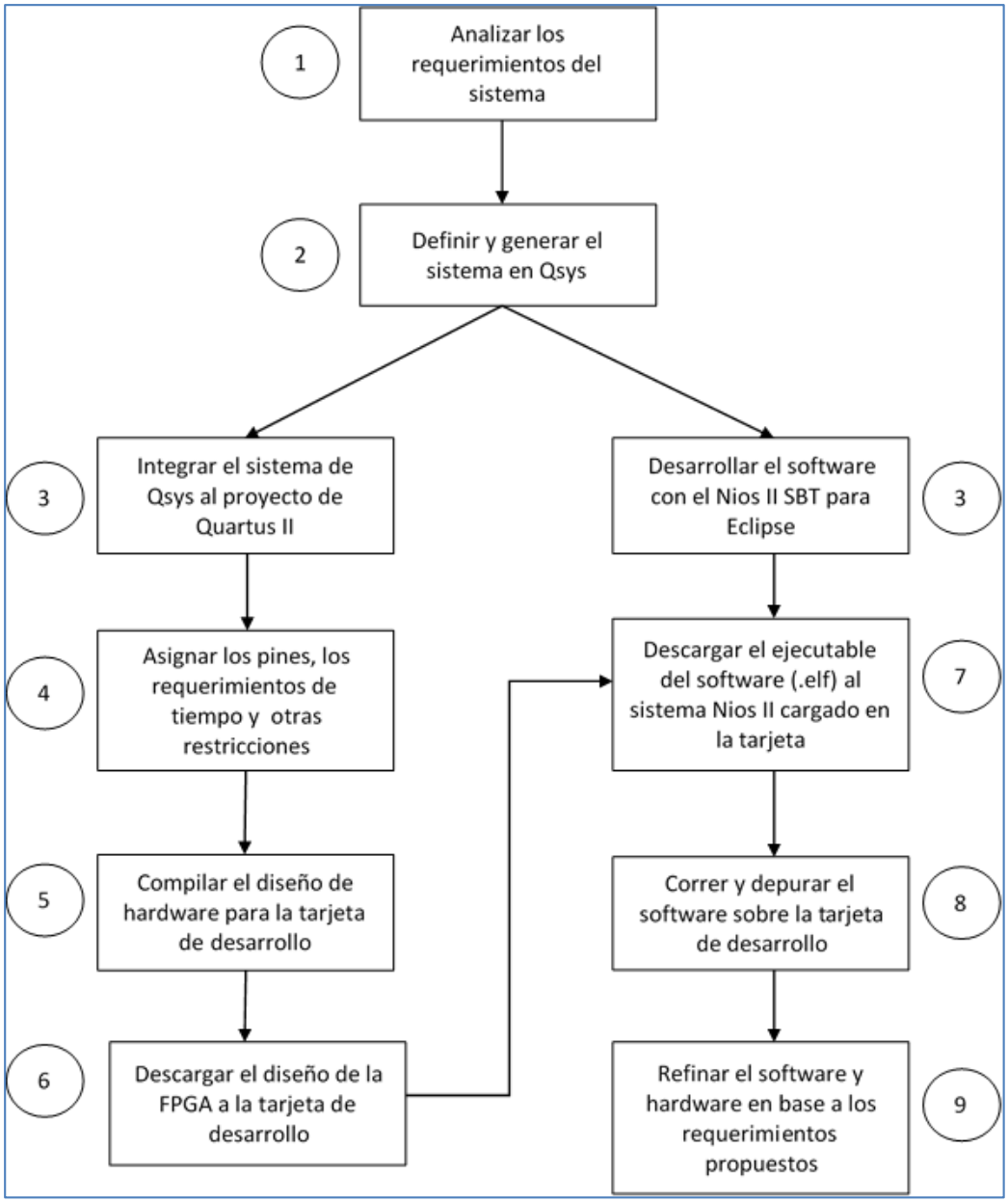


Figura 2.8 Flujo de desarrollo para la creación de un sistema Nios II [13]

2.5 PERIFÉRICOS OFF-CHIP DEL SISTEMA NIOS II INTEGRADOS EN LA TARJETA DE0-NANO.

Se describen las bases teóricas de los periféricos incorporados en la tarjeta DE0-Nano empleados en el desarrollo de los sistemas Nios II de las prácticas elaboradas en el presente proyecto.

2.5.1 Chip SDRAM ISSI IS42S16160B

Es una memoria DRAM (Dynamic Random Access Memory) sincrónica, lo que quiere decir que a diferencia de las clásicas DRAMs espera por una señal de reloj antes de responder a señales de entrada de control. Puede almacenar 256Mb (32MB organizados: 4M x 16 x 4 bancos) y alcanza altas velocidades de transferencia de datos. Cada uno de los 4 bancos están organizados en 8192 filas (2^13), 512 columnas (2^9) por 16 bits. Todas las señales de entrada y salida están referidas a los flancos de subida del reloj de entrada. En lo que respecta a la alimentación de voltaje utiliza estándar 3.3V LVTTL [14].

LA SDRAM incrustada en la tarjeta DE0-nano es de 16 bits mientras que el procesador Nios II es de 32 bits, por lo tanto el ancho de dato del esclavo es menor que el del maestro, debido a esto Qsys inserta lógica para traducir entre ellos, esta lógica adicional, no es lo más recomendable debido a que puede provocar mayor latencia entre cada transacción; sin embargo los sistemas que se desarrollan no exigen cumplir a cabalidad esta sugerencia [15].

Los accesos de lectura y escritura a la SDRAM se realizan de forma intermitente; el acceso empieza en una locación seleccionada y continua hasta el número programado de localidades. El acceso empieza con el registro del comando ACTIVE el mismo que es seguido por comando de escritura (WRITE) o lectura (READ). Principalmente la SDRAM debe ser inicializada, para ello es requerido un retraso de 200us, luego de ello el comando PRECHARGE debe ser aplicado. Todos los bancos deben ser

precargados. Finalmente para que la SDRAM esté lista indicarle el específico modo de operación a realizar deben desarrollarse al menos 8 ciclos de AUTO REFRESH [14].

En el párrafo previo únicamente se explicó el proceso de inicialización de la SDRAM, este proceso en conjunto con los de carga, lectura y escritura sería una tarea muy demandante es por ello que Qsys provee un controlador de SDRAM entre sus componentes.

El SDRAM Controller provee todas las señales necesarias para comunicarse con el off-chip SDRAM excepto la señal de clock, en la Figura 2.9 se observan dichas señales y en la Tabla 2.3 se indican los ajustes de tiempo del controlador comparando los valores por defecto y los que se deben ajustar de acuerdo a la SDRAM propia de la tarjeta DE0-Nano. La señal de clock debe satisfacer los requerimientos de la memoria en cuanto al desplazamiento del reloj, para una apropiada operación del chip SDRAM es necesario que la entrada de clock del chip adelante por 3 nanosegundos al reloj del sistema Nios II. Este reloj puede ser provisto por un PLL (Phase-Locked Loop) creado manualmente con la herramienta MegaWizard plug-in o creado automáticamente al usar el componente Clock Signals [16].

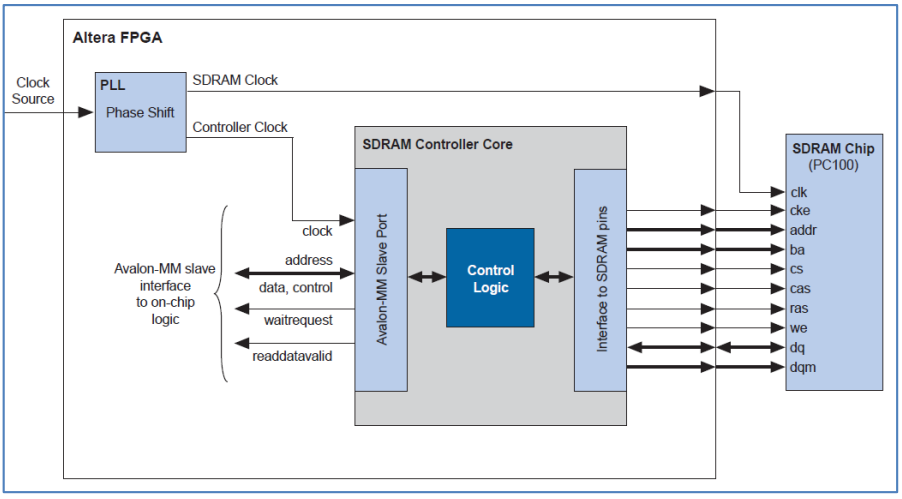


Figura 2.9 Diagrama de bloques del controlador SDRAM de Qsys [9].

Ajuste	Valor por defecto	Valor de la SDRAM IS42S16160B	Unidad	Descripción
CAS latency cycles	3	3	[ciclos]	Retardo entre el comando de lectura y la salida de datos. También puede configurarse con 2 ciclos
Initialization refresh cycles	2	8	[ciclos]	Cuantos ciclos de reloj debe realizar como parte de la secuencia de inicialización después de un reset.
Issue one refresh command every:	15.625	7.8125	[us]	Especifica que tan seguido el controlador refresca a la SDRAM. La IS42S16160B requiere 8192 comandos de refrescar cada 64 ms. Cada comando se da en 64ms / 8192 = 7.8125us
Delay after powerup before initialization	100	200	[us]	Tiempo de retardo para estabilizar el reloj luego del encendido y empezar proceso de inicialización de la SDRAM
t_rfc	70	70	[ns]	Periodo de auto refrescar
t_rp	20	20	[ns]	Periodo de precargar
t_rcd	20	20	[ns]	Retardo desde el commando ACTIVE hasta los commandos READ o WRITE
t_ac	17	5.4	[ns]	Tiempo de acceso del flanco de reloj
t_wr	14	14	[ns]	Recuperación de escritura si un comando explícito de precarga es emitido

Tabla 2.3 Ajustes de tiempo del componente SDRAM Controller de Qsys [14], [16].

2.5.2 Chip ADC128S022

Es el convertidor analógico-digital de bajo consumo de potencia que se encuentra integrado en la tarjeta DE0-Nano. Tiene ocho canales de entradas analógicas, cada una de las mismas se convierten a señales digitales de 12 bits. Su tasa de rendimiento esta entre 50 – 200 ksps (kilo samples per second); lo que quiere decir que puede tomar desde 50 hasta 200 muestras por segundo. La señal digital de salida utiliza el tipo de comunicación serial SPI que es soportado por la FPGA Cyclone IV. Entre las aplicaciones que se le pueden dar a este convertidor están: instrumentación médica, comunicaciones móviles e instrumentación y control de sistemas.

La tarjeta DE0-Nano cuenta con un cabezal de 2x13 pines, que se muestra en la Figura 2.10, el mismo provee acceso a las ocho entradas analógicas del ADC128S022.

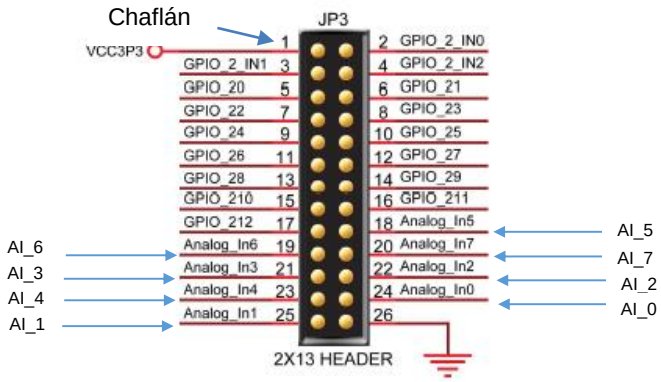


Figura 2.10 Ubicación de las entradas analógicas en el cabezal de pines 2x13 (vista superior)

Máxima entrada analógica.- Para evitar el daño de la tarjeta DE0-Nano cualquier voltaje provisto al ADC por medio del cabezal de pines de 2x13 no debe exceder los 3.3V.

En la Tabla 2.4 se visualiza como se debería realizar las conexiones entre el chip ADC y la FPGA, en la columna tipo se indica si es entrada (I) o salida

(O) con respecto a la FPGA. Ejemplo: En Quartus II se le dará el nombre ADC_SDAT, tipo entrada y se le asigna el pin_A9 estableciendo así la conexión con el pin DOUT del chip ADC, por supuesto desde el punto de vista del chip este pin es de salida.

Tipo	***Nombre del pin en el chip ADC***	***Nombre de la señal en la FPGA***	***No. Pin de la FPGA***	***Descripción de los pines del chip ADC128S022***
O	CS_n	ADC_CS_N	PIN_A10	*En los flancos de bajada de CS_n comienza la conversión y se mantiene mientras este abajo*
O	DIN	ADC_SADDR	PIN_B10	*Entrada de datos digitales. El registro de control del ADC se carga en los flancos de subida del SCLK*
I	DOUT	ADC_SDAT	PIN_A9	*Salida de datos digitales. Está sincronizado con los flancos de bajada del SCLK*
O	SCLK	ADC_SCLK	PIN_B14	*Entrada de clock digital. Para garantizar el desempeño la frecuencia debe estar entre 0.8 MHz a 3.2 MHz.*

Tabla 2.4 Asignación de pines para el ADC

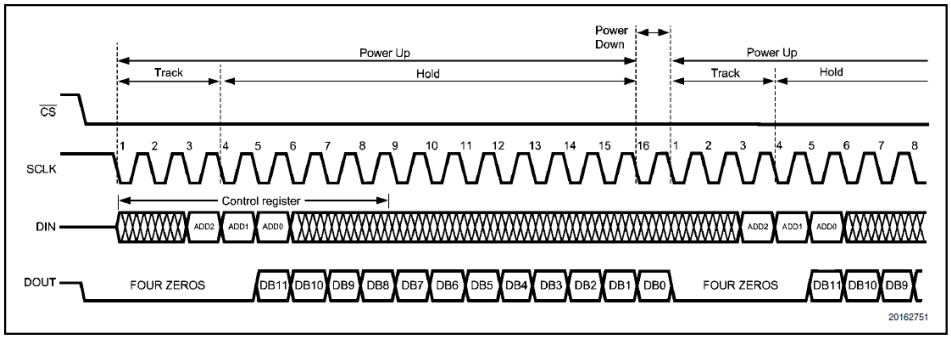


Figura 2.11 Diagrama de tiempo de operación del ADC128S022

Como se observa en la Figura 2.11 para llevar a cabo la conversión de un dato analógico a un dato digital al chip ADC128S022 le toma 16 flancos de reloj del ADC (SCLK). El inicio de la conversión se da desde que la entrada $\overline{CS}$ recibe un flanco negativo y se mantiene así, se lee además el registro de control sobre el próximo canal a leer y envía por DOUT los 12 bits de salida que representan a la entrada analógica en binario. Con una frecuencia de 3.2MHz se obtendría (3.2M/16) Ksps; es decir con un reloj a esa frecuencia se toman 200000 muestras por segundo.

Por defecto, al iniciar el proceso de conversión, el convertidor toma el dato del canal 0 y para la siguiente conversión toma el dato del canal correspondiente de acuerdo a lo que se envíe por la entrada (con respecto al chip ADC) DIN, que recibe tres bits, desde 000 hasta 111 para representar los 8 canales disponibles.

El chip ADC es controlado por el componente DE0-Nano ADC Controller, el mismo que se encuentra en la sección University Program de la librería de componentes en Qsys. Este componente provee una interfaz de registro asignado en memoria cuya dirección se obtiene al hacer la asignación de direcciones base en Qsys.

Si se considera s el caso particular de un Base Address = 0x00009000 y End Address = 0x0000901F, la distribución de la memoria en base a los ocho canales del ADC sería como se muestra en la Tabla 2.5.

ADDRESS	31...12	11...0	Registro de los canales
0X00009000	Sin usar	Valor del canal 0	Registro del canal 0 / Update
0X00009004	Sin usar	Valor del canal 1	Registro del canal 1/ Auto-Update
0X00009008	Sin usar	Valor del canal 2	Registro del canal 2
0X0000900C	Sin usar	Valor del canal 3	Registro del canal 3
. .	. .	. .	. .
0X0000901C	Sin usar	Valor del canal 7	Registro del canal 7

Tabla 2.5 Distribución de memoria para el chip ADC

Para tomar las lecturas y convertirlas se debe escribir 1 en el bit 0 del registro del canal 1/ Update, a su vez se puede configurar el controlador para continuamente actualizar mediante la escritura en el bit 0 del registro del canal 1 / Auto-Update.

Se accede al valor adquirido más reciente de cada canal mediante la lectura del registro de dicho canal. La escritura sobre algún registro diferente al de Update o Auto-Update es ignorado.

Los niveles de tensión a conectar en las entradas del ADC no deben superar los 3.3V, es por ello que para la práctica se debe realizar un divisor de tensión con una alimentación de ese valor máximo de voltaje. En aplicaciones reales, los valores a medir son superiores, para ello deberá acondicionarse la señal mediante etapas de acoplamiento y amplificación.

2.5.3 Chip Acelerómetro ADI ADXL345

Es un pequeño acelerómetro de 3 ejes, con bajo consumo de energía y alta resolución de medida (hasta 13 bits). Se puede conectar a través de una interfaz de 3 cables SPI o de dos cables I2C. En la tarjeta DE0-Nano el acelerómetro está conectado a la FPGA mediante el arreglo SPI de 3 hilos,

la conexión entre ambos se visualiza en la Figura 2.12. Entre sus aplicaciones se pueden mencionar: instrumentación médica e industrial, dispositivos de navegación personal, protección de discos duros, como sensor de inclinación permitiendo medir variaciones menores a 1°.

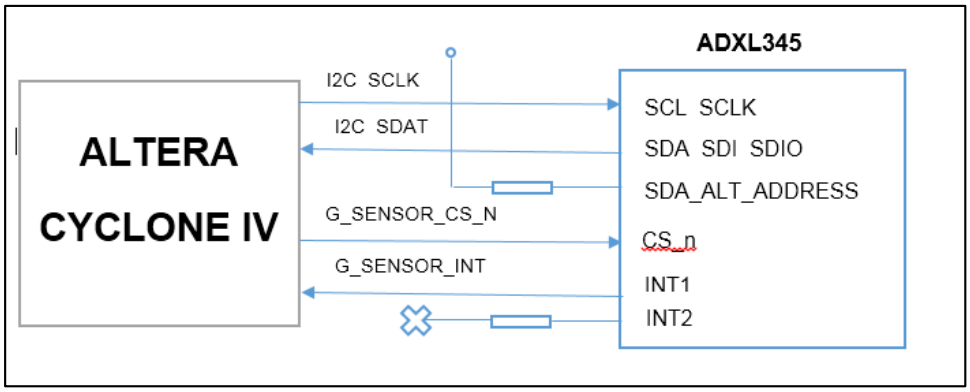


Figura 2.12 Conexiones entre el chip ADXL345 y la FPGA Cyclone IV

En la Tabla 2.6 se visualiza como se debería realizar las conexiones entre el chip ADXL345 y la FPGA, en la columna tipo se indica si es entrada (I) o salida (O) con respecto a la FPGA. Ejemplo: En Quartus II se le dará el nombre I2C_SCLK, tipo salida y se le asigna el pin_F2 estableciendo así la conexión con el pin SCL_SCLK del chip ADXL345, por supuesto desde el punto de vista del chip este pin es de entrada.

Tipo	*Nombre del pin en el chip ADXL345*	*Nombre de la señal en la FPGA*	*No. Pin de la FPGA*	*Descripción de los pines del chip ADC128S022*
O	SCL_SCLK	I2C_SCLK	PIN_F2	
I	SDA_SDI_SDIO	I2C_SDAT	PIN_F1	
O	CS_n	G_SENSOR_CS_N	PIN_G5	*Selecciona el modo de comunicación.* $\overline{CS} = 1$ *para I2C y* $\overline{CS} = 0$ *para SPI*
I	INT1	G_SENSOR_INT	PIN_M2	

Tabla 2.6 Asignación de pines para el Acelerómetro

En la Tabla 2.7 se presentan las opciones de resolución de salida que dependen de la selección del rango g, para aplicaciones de medición de inclinación el rango óptimo es ±2 g. La Figura 2.13 muestra la salida del acelerómetro con respecto a la posición.

Resolución de salida		**Cada eje**
Todos los rangos g	10 bits de resolución	10 bits
Rango ±2 g	Resolución full	10 bits
Rango ±4 g	Resolución full	11 bits
Rango ±8 g	Resolución full	12 bits
Rango ±16 g	Resolución full	13 bits

Tabla 2.7 Resoluciones de salida disponibles

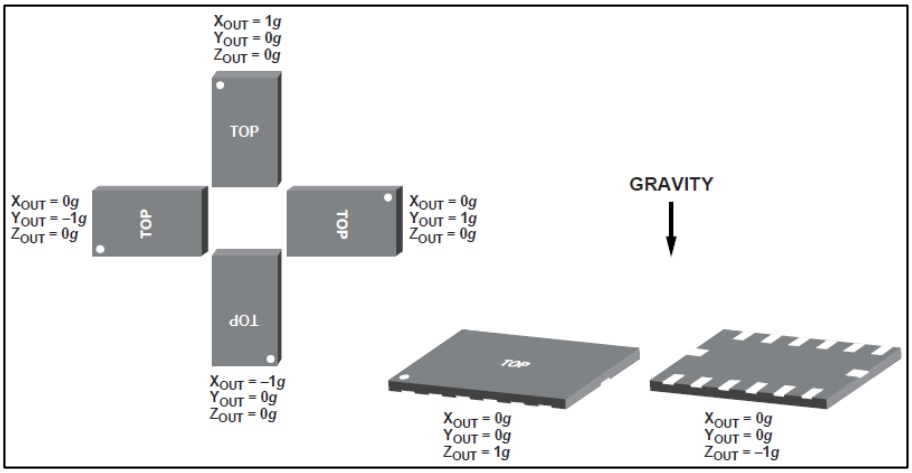


Figura 2.13 Respuesta de salida vs orientación respecto a la gravedad

El componente de Qsys, Accelerometer SPI mode, tiene dos registros de 8 bits, llamados address y data. Los mismos son mapeados a direcciones específicas una vez que al componente se le asigna las direcciones base en Qsys, estos registros se muestran en la Tabla 2.8.

Por medio de estos registros el procesador puede especificar el modo de operación del acelerómetro y leer sus valores actuales.

OFFSET EN BYTES	NOMBRE DEL REGISTRO	LECTURA/ESCRITURA	7…6	5…0
0	address	E	0	Dirección
1	data	L/E	Dato del registro	

Tabla 2.8 Registros del componente Accelerometer SPI mode de Qsys

Para acceder a las medidas de los tres ejes se cuenta con dos pares de registros por eje, uno para los 8 bits más significativos (MSB) y otro para los LSB, para ello se debe escribir la dirección correspondiente en los 6 bits menos significativos del registro address. Los registros que contienen las medidas por eje se observan en la Tabla 2.9.

EJE	DATO	Dirección para acceder
X	DATA X0 (8 bits menos significativos)	0X32
	DATA X1 (8 bits más significativos)	0X33
Y	DATA Y0 (8 bits menos significativos)	0X34
	DATA Y1 (8 bits más significativos)	0X35
Z	DATA Z0 (8 bits menos significativos)	0X36
	DATA Z1 (8 bits más significativos)	0X37

Tabla 2.9 Direcciones de las medidas del acelerómetro por eje.

Cada vez que el diseño de hardware es cargado a la tarjeta, se realiza una auto inicialización del componente Accelerometer SPI mode, que tiene una configuración automática que causa el desarrollo continuo de las mediciones y permite la lectura de los resultados en las parejas de registros de X, Y y Z usando una resolución de ±2 g.

Los valores obtenidos se los puede presentar en decimal, simplemente almacenándolos en una variable de ese tipo; pero para obtener su valor de aceleración gravitacional se aplica la ecuación 2.1:

$$valor_{gravitacional} = \frac{valor_{decimal}}{511} * 2 \qquad (2.1)$$

CAPÍTULO 3

3. APORTE DE CADA PRÁCTICA AL APRENDIZAJE DE SISTEMAS EMBEBIDOS BASADOS EN NIOS II.

3.1 COMPOSICIÓN GENERAL DE LAS PRÁCTICAS

El objetivo general de la propuesta es el diseño e implementación de prácticas con *Qsys*, *Nios II Software Build Tools for Eclipse* y la tarjeta DE0-Nano, para el aprendizaje de sistemas embebidos basados en el procesador Nios® II, por lo tanto para llegar a este objetivo se proponen 6 prácticas y cada una de ellas cuenta con los archivos y carpetas que se presentan en la Tabla 3.1 para dar soporte al estudiante en el aprendizaje. Las prácticas están diseñadas para ser realizadas en promedio entre una hora y media a dos horas sin considerar el tiempo de lectura de la fundamentación teórica de las mismas, se propone que sean realizadas en grupos de dos estudiantes.

Nombre del archivo o carpeta	**Descripción / Contenido**
PRÁCTICA_<Nombre de la práctica>	• Objetivos de la práctica • Requisitos mínimos, como haber realizado prácticas previas y comprensión de la fundamentación teórica. • Explicación general de la práctica • Desarrollo paso a paso de la práctica con incorporación de capturas de pantalla y mención a prácticas previas para reforzar alguna parte en el flujo del diseño que se requiera.

FUNDAMENTACIÓN_TEÓRICA_PRÁCTICA_ <Nombre de la práctica>	• Se describen todos los principios de las herramientas y/o componentes a utilizar. • Explicaciones más amplias de los ajustes que se hacen en el diseño de hardware y software de la práctica. • Menciona los manuales a los que se puede recurrir para ampliar conocimientos al menos en el campo comprendido por la práctica en cuestión.
MANUALES_PRÁCTICA_<Nombre de la práctica>	Contiene los manuales de donde se ha realizado la recopilación de información para la fundamentación teórica y permite ampliar la lectura sobre el tema tratado en la práctica.
Archivos adicionales	**Práctica 1.** Contiene un archivo que explica la creación de proyectos en Quartus II en caso que el estudiante requiera recordar o sea nuevo en el tema.

Tabla 3.1 Descripción de los archivos y carpetas de las prácticas

En la Figura 3.1 se muestra el contenido de la carpeta de la práctica 1, donde se observan: la carpeta y todos los archivos descritos en la tabla anterior de esta práctica, de igual forma se tienen para las demás.

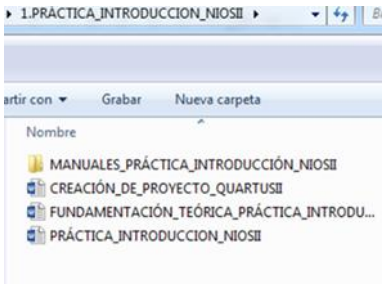


Figura 3.1 Contenido de la carpeta de la Práctica 1.

3.2 FLUJO DEL APRENDIZAJE DE SISTEMAS EMBEBIDOS SEGÚN EL AVANCE DE LAS PRÁCTICAS

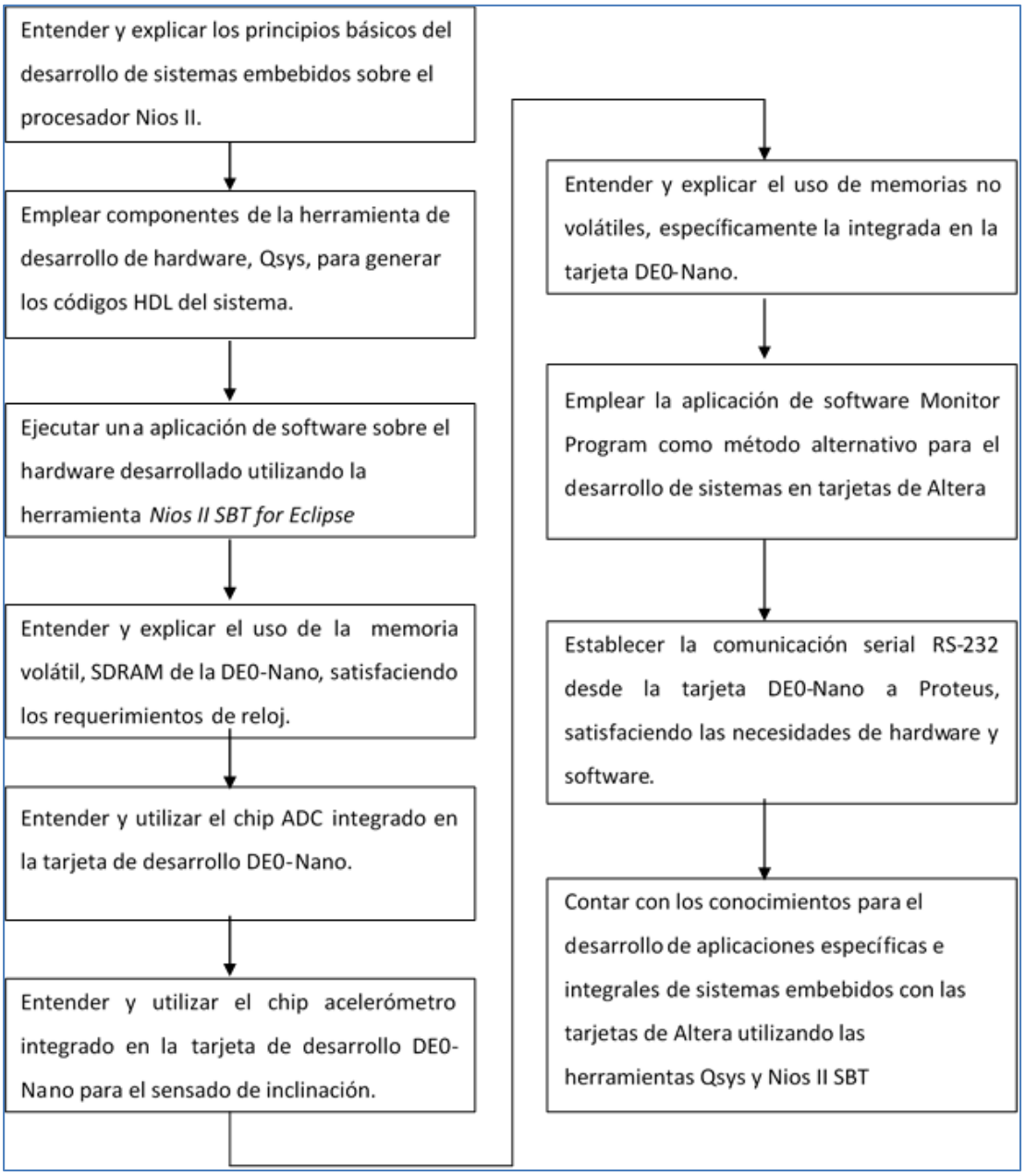


Figura 3.2 Diagrama de flujo del aprendizaje de sistemas embebidos

A continuación se explica cada objetivo del diagrama de flujo del aprendizaje de sistemas embebidos de la Figura 3.2 y mediante que práctica es cubierto.

3.2.1 Entender y explicar los principios básicos del desarrollo de sistemas embebidos sobre el procesador Nios II.

Este objetivo será cubierto con todas las prácticas al ir adquiriendo nuevos conocimientos en el desarrollo de sistemas embebidos; pero es en la práctica 1 que se presentan las bases del flujo de diseño, para ello la fundamentación teórica presenta las herramientas y componentes a utilizar y en el desarrollo de esta práctica se interactúa por primera ocasión con Qsys y Nios II SBT para el diseño de hardware y software respectivamente.

3.2.2 Emplear componentes de la herramienta de desarrollo de hardware, Qsys, para generar los códigos HDL del sistema.

En la práctica 1 se explica detalladamente la selección de los componentes requeridos del sistema, especificación de sus parámetros, interconexión de los mismos mediante una matriz de conexiones que ofrece la herramienta y asignación de direcciones de memoria de cada parte del sistema Nios II. Este objetivo es reforzado en el desarrollo de las demás prácticas porque en cada una de ellas se crean nuevos sistemas de hardware mediante Qsys.

3.2.3 Ejecutar una aplicación de software sobre el hardware desarrollado utilizando la herramienta Nios II SBT for Eclipse

Para alcanzar este objetivo en la práctica 1 se detalla paso a paso como correr un programa básico creado en Nios II Software Build for Eclipse sobre el diseño de hardware desarrollado, mediante el uso de una plantilla de proyecto vacío, al mismo se le añade un archivo de código fuente de un contador binario, se ejecuta y visualiza el funcionamiento del mismo en los 8 LEDs incorporados en la tarjeta DE0-Nano, mediante la correcta asignación a la dirección base de los LEDs. Este objetivo es reforzado en

el desarrollo de las demás prácticas porque en cada una de ellas se corren nuevas aplicaciones de software sobre el sistema de hardware.

3.2.4 Entender y explicar el uso de la memoria volátil, SDRAM de la DE0-Nano, satisfaciendo los requerimientos de reloj.

Este objetivo se cubre mediante el desarrollo de la práctica 2, la importancia de esta práctica de debe a que para aplicaciones comunes se requieren mayores cantidades de memoria para el programa y para ello se usan memorias externas como la SDRAM de la DE0-Nano. Mediante el componente SDRAM Controller se crea la interfaz que provee todas las señales necesarias para comunicarse con el off-chip SDRAM. En algunas prácticas posteriores se utilizará para evitar desbordes de memorias. En esta práctica se utiliza el componente Clock Signals de la librería University Program con configuraciones específicas para satisfacer las necesidades de tiempo de la memoria añadida. Como parte de la segunda práctica y para aprovechar la capacidad de memoria de la SDRAM se presenta el control por software de la LCD 16x2, mediante el acceso a los registros de memoria asignados en la parte de creación de hardware.

3.2.5 Entender y utilizar el chip ADC integrado en la tarjeta de desarrollo DE0-Nano.

La práctica 3 presenta el material necesario para instruirse en la adquisición de datos de los canales del chip convertidor ADC integrado en la tarjeta DE0-Nano. Al crear el sistema de hardware en Qsys, entre los componentes utilizados se encuentra el DE0-Nano ADC Controller, el mismo que establece una interfaz entre el procesador y el chip ADC128S002 de la tarjeta DE0-Nano. Se evidencia el correcto desarrollo de la práctica mostrando en la consola el voltaje del canal seleccionado con los interruptores de la tarjeta y con la visualización en los 8 LEDs incorporados en la tarjeta DE0-Nano de los 8 bits más significativos de la conversión del ADC, para ello se realiza una aplicación de software con Nios II SBT y se muestra y explica el acceso a los registros del ADC.

3.2.6 Entender y utilizar el chip acelerómetro integrado en la tarjeta de desarrollo DE0-Nano para el sensado de inclinación.

Este objetivo se cubre en la práctica 4, en la cual entre los componentes que se añaden al sistema de hardware mediante Qsys se encuentra el componente de propiedad intelectual IP: Acelerometer SPI Mode de la librería University Program que se utiliza para el control del chip ADXL345, un acelerómetro de tres ejes contenido en la tarjeta de desarrollo DE0-Nano. Sobre el diseño de hardware desarrollado en la práctica 4 se corre un programa creado en Nios II SBT for Eclipse, que permite mostrar en consola los valores medidos por el acelerómetro en el eje X y los ocho bits menos significativos de la lectura se muestran en los leds de la tarjeta. Se explica además como se da la lectura de los otros ejes (Y, Z), para ello se explica el acceso a los registros del controlador del acelerómetro utilizado.

3.2.7 Entender y explicar el uso de memorias no volátiles, específicamente la integrada en la tarjeta DE0-Nano.

Mediante el desarrollo de la práctica 5 se alcanza este objetivo, en esta práctica se crea un sistema embebido de arranque mediante el uso de la memoria flash Spansion S25FL064P, integrada en la tarjeta DE0-Nano.

3.2.8 Emplear la aplicación de software Monitor Program como método alternativo para el desarrollo de sistemas en tarjetas de Altera

En la práctica 5 se cubre este objetivo al emplear tanto el Nios II SBT for Eclipse y la aplicación provista por Altera, Monitor Program, para compilar y descargar en la tarjeta el programa a correr sobre el hardware diseñado en Qsys.

3.2.9 Establecer la comunicación serial RS-232 desde la tarjeta DE0-Nano a Proteus, satisfaciendo las necesidades de hardware y software.

Este objetivo se cubre con el desarrollo de la práctica 6, entre los componentes añadidos al sistema de hardware con Qsys se encuentra el

componente UART (RS-232 Serial Port), en el cual se configuran los parámetros de la comunicación serial desde el lado de la tarjeta DE0-Nano. Sobre el diseño de hardware desarrollado se corre un programa creado en Nios II SBT for Eclipse, que permite mostrar en consola los caracteres recibidos desde proteus y enviar caracteres desde la tarjeta DE0- Nano, que serán visualizados en un terminal virtual desde el otro extremo de la comunicación serial.

3.2.10 Contar con los conocimientos para el desarrollo de aplicaciones específicas e integrales de sistemas embebidos con las tarjetas de Altera utilizando las herramientas Qsys y Nios II SBT.

Este objetivo es alcanzado una vez que se han concluido todas las prácticas y los estudiantes están en la capacidad de plantearse o resolver aplicaciones que se les solicite en forma de proyectos en los cuales deban hacer uso de las herramientas aprendidas para el desarrollo de sistemas embebidos ya sea con los componentes de la tarjeta o sino periféricos externos al acceder desde software a los registros de memoria que se les asigne en la etapa de creación de hardware en el flujo del diseño.

3.3 COSTOS DE INVERSIÓN

Para la correcta realización de las prácticas los estudiantes deben de contar con los equipos y elementos necesarios, es por ello que se exponen en la Tabla 3.2 los costos de los mismos. Cabe señalar que las cantidades están consideradas para un laboratorio que de acogida a 10 estudiantes durante la práctica y se trabaje en parejas.

Artículo	**Costo unitario ($)**	**Cantidad**	**Costo Total ($)**
Tarjeta de desarrollo DE0-Nano	61,00	10	610,00*
Software Quartus II + Qsys	2995,00	1	2995,00*
Subtotal_Altera (1)			3605,00*
Resistencias 100 ohms ½ w	0,03	100	3,00
Potenciómetros (Tipo trimmer)	0,10	50	5,00
CI MAX232	0,50	20	10,00
Capacitores (1uF)	0,10	100	10,00
LCD 16 X 2	5,00	10	50,00
Convertidor RS232 a USB	7,00	10	70,00
Metros de cable UTP	0,30	50	15,00
Subtotal_elementos (2)			163,00
TOTAL [(1)+(2)]			3768,00

Tabla 3.2 Detalle de los costos de inversión

El valor Subtotal_Altera (1), que se indica en la Tabla 3.2, corresponde a un rubro referencial considerando los precios que Altera en su página web expone, sin embargo, este fabricante menciona múltiples beneficios cuando son instituciones académicas las que desean acceder a sus productos con fines educativos, para ello

se debe contactar con ellos un representante de la institución educativa y poder acceder a precios preferenciales.

En el caso de la ESPOL, ya se cuenta tanto con las tarjetas de desarrollo, cerca de 25 unidades de las mismas y además una licencia del software con fecha de expiración al 10 de julio del 2025, es por ello que implementar estas prácticas únicamente requeriría la inversión de elementos, el valor Subtotal_elementos (2), que se muestra en la Tabla 3.2 el mismo que alcanza la simbólica cantidad de $163.

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

1. La metodología de aprendizaje distribuida y secuencial en las que se integran las prácticas permitirán al estudiante que adquiera los conocimientos de cada una de ellas y las refuerce en la práctica siguiente.

2. El enfoque dado a las prácticas en priorizar sobre el entendimiento del flujo de desarrollo del sistema y acceso a los componentes que lo conformen mediante la lectura y escritura a sus direcciones permitirán al estudiante realizar sus propias aplicaciones, de mayor alcance y más específicas.

3. Las prácticas están fundamentadas sobre el procesador Nios II y utilizan únicamente la tarjeta DE0-Nano para su implementación; sin embargo el aprendizaje que el estudiante obtendrá en dichas prácticas le permitirá también crear y descargar sus propios sistemas embebidos sobre otras tarjetas de Altera con mayores recursos y que le ofrecerán nuevos retos en la línea de aprendizaje.

4. Las prácticas presentadas ofrecen una base sólida para que sobre las mismas se puedan trabajar proyectos que incorporen lógica adicional a los sistemas; pero para asegurar la comprensión y posibilidad de realizar modificaciones coherentemente se recomienda que todas las prácticas hayan sido previamente concluidas satisfactoriamente.

5. Dada la situación actual de la ESPOL, en cuanto a las tarjetas DE0-Nano con las que ya cuenta, al igual que la licencia del software requerido para cerca de 9 años más, la implementación de esta guía de prácticas y proyectos que puedan surgir basados en las mismas se puede realizar de forma casi inmediata.

RECOMENDACIONES

1. Se recomienda la adquisición de tarjetas con mayores recursos para aspirar a aplicaciones más exigentes y no únicamente de Altera sino también de Xilinx para que los estudiantes cuenten con el conocimiento de desarrollo de sistemas embebidos basados en FPGAs de los dos fabricantes que acaparan el 90% de ese mercado.

2. Se recomienda que una vez concluidas las prácticas, los estudiantes deban realizar un único proyecto; pero con un alcance tal que permita evidenciar el aprendizaje obtenido.

BIBLIOGRAFÍA

[1] Noergaard, (2005). Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers, Section I [online]. Disponible en: http://booksite.elsevier.com/samplechapters/9780750677929/9780750677929.PDF

[2] Bisdounis, (2007, septiembre). Embedded systems & applications and system-on-chip design [online]. Disponible en: http://bisdounis.ele.teiwest.gr/files/P03_en.pdf

[3] National Instruments, (2011, diciembre). Understanding Parallel Hardware: Multiprocessors, Hyperthreading, Dual-Core, Multicore and FPGAs. Disponible en: http://www.ni.com/tutorial/6097/en/

[4] National Instruments, NI LabVIEW for CompactRIO Developer's Guide [online]. Disponible en: http://www.ni.com/pdf/products/us/fullcriodevguide.pdf

[5] Terasic Technologies Inc, (2012, diciembre). DE0-Nano User Manual [online]. Disponible en: http://www.altera.com/literature/ug/DE0_Nano_User_Manual_v1.9.pdf

[6] Altera, (2015, abril). Nios II Classic Processor Reference Guide [online]. Disponible en: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/nios2/n2cpu_nii5v1.pdf

[7] Altera, (2015, diciembre). Avalon Interface Specifications [online]. Disponible en: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/mnl_avalon_spec.pdf

[8] Altera, (2012, octubre). Introduction to the Altera Qsys System Integration Tool [online]. Disponible en: ftp://ftp.altera.com/up/pub/Altera_Material/12.1/Tutorials/Introduction_to_the_Altera_Qsys_Tool.pdf

[9] Altera, (2015, diciembre). Embedded IP Peripherals User Guide [online]. Disponible en: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_embedded_ip.pdf

[10] Altera, (2013, mayo). Quartus II Handbook Version 13.0 [online]. Disponible en: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/qts/archives/quartusii_handbook_archive_130.pdf

[11] Altera, (2015, mayo). Nios II Classic Software Developer's Handbook [online]. Disponible en: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/nios2/n2sw_nii5v2.pdf

[12] Wikipedia, (2015, diciembre). Eclipse (software) [online]. Disponible en: https://es.wikipedia.org/wiki/Eclipse_%28software%29

[13] Altera, (2011, mayo). Nios II Hardware Development Tutorial [online]. Disponible en: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/tt/tt_nios2_hardware_tutorial.pdf

[14] ISSI, (2008, septiembre). 256-MBIT SYNCHRONOUS DRAM [online]. Disponible en: http://www.issi.com/WW/pdf/42S83200B-16160B.pdf

[15] Altera, (2015, diciembre). Embedded Design Handbook [online]. Disponible en: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/nios2/edh_ed_handbook.pdf

[16] Altera, (2012, mayo). Using the SDRAM on Altera's DE0-Nano Board with VHDL Designs [online]. Disponible en: ftp://ftp.altera.com/up/pub/Altera_Material/12.0/Tutorials/VHDL/DE0-Nano/Using_the_SDRAM.pdf

ANEXOS

Anexo 1: Guía de prácticas para el aprendizaje de sistemas embebidos

PRÁCTICA # 1

TEMA: *Introducción al diseño de hardware y creación de software para un sistema basado en el procesador Nios II utilizando la tarjeta de desarrollo DE0-Nano.*

1. Objetivos:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Entender y explicar los principios básicos del desarrollo de sistemas embebidos sobre el procesador Nios II.
- Emplear componentes básicos de la herramienta de desarrollo de hardware, *Qsys,* para generar los códigos HDL del sistema.
- Ejecutar una aplicación de software básico sobre el hardware desarrollado utilizando plantillas de la herramienta *'Nios II Software Build Tools for Eclipse'.*

2. Requisitos mínimos:

- Tener instalado Quartus II y Nios II EDS (versión 13.0).
- Conocimientos básicos de: Quartus II y programación en VHDL y C.
- Lectura y comprensión del archivo:
  'FUNDAMENTACIÓN_TEÓRICA_PRÁCTICA_INTRODUCCIÓN_NIOSII'

3. Breve explicación de la práctica:

Se desarrolla un sistema de hardware que incluye un procesador embebido de Altera Nios II en conjunto con otros módulos *(véase Figura 1)*, conectándose por medio de

una interconexión llamada *Avalon switch fabric.* De esta manera se forma un sistema que luego se implementa en el chip FPGA de la tarjeta DE0-Nano utilizando el programa *Quartus II*.

Todas las partes del sistema basado en el procesador Nios II, al cual llamaremos sistema Nios II, son definidas usando lenguaje de descripción de hardware, como verilog o VHDL; pese a que se podría escribir el código para implementar cada parte, esto consumiría mucho tiempo, por ello esta práctica enseña cómo utilizar la herramienta de *Qsys* para implementar el sistema deseado mediante la selección de los componentes requeridos, especificación de sus parámetros, interconexión de los mismos mediante una matriz de conexiones que ofrece la herramienta y asignación de direcciones de memoria de cada parte del sistema Nios II.

Sobre el diseño de hardware desarrollado se corre un programa básico creado *en Nios II Software Build Tools for Eclipse,* mediante el uso de una plantilla de proyecto vacío, al mismo se le añade un archivo de código fuente de un contador binario, se ejecuta y visualiza el funcionamiento del mismo en los 8 LEDs incorporados en la tarjeta DE0-Nano.

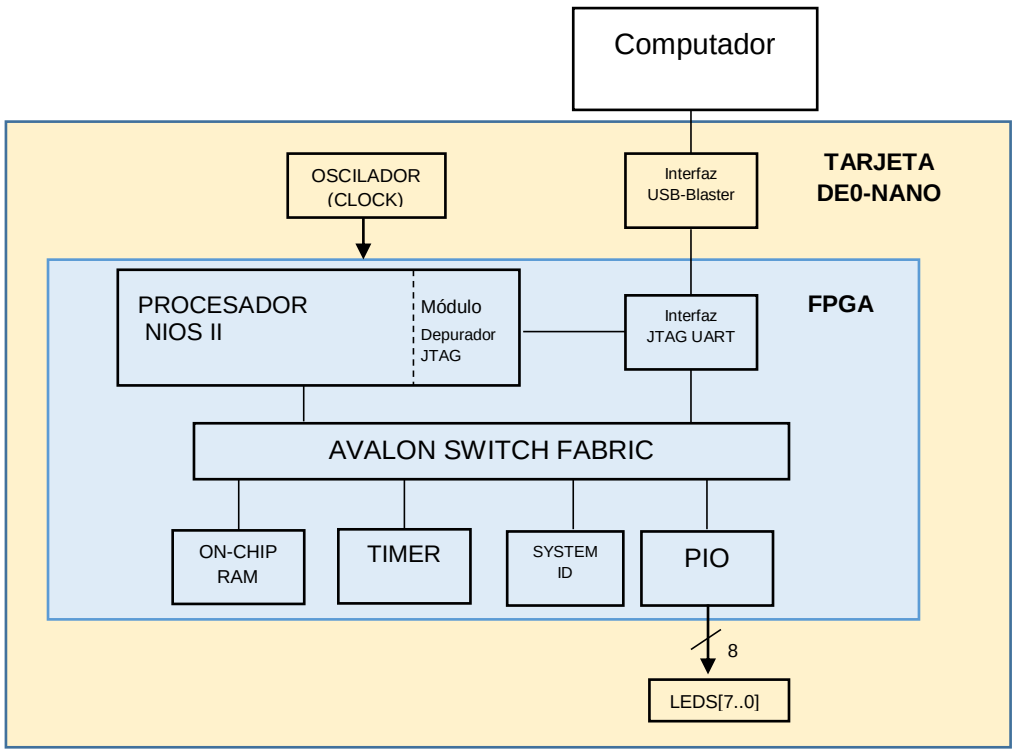


Figura 1. Sistema Nios II a implementar en la Práctica#1

4. Desarrollo de la práctica:

1. Abrir el programa Quartus II y crear un proyecto nuevo (*Nombre del proyecto sugerido: practica_1)* con la familia *Cyclone IV E, dispositivo EP4CE22F17C6. (En caso de necesitar mayor detalle en la creación de proyectos en Quartus II refiérase al archivo CREACIÓN_DE_PROYECTO_QUARTUSII contenido en la carpeta de la presente práctica*.)

   ***Nota**. Durante la creación del proyecto → En la ventana EDA tool settings, seleccione <None> en todas las opciones de la columna Tool Name, debido a que no se utilizarán herramientas de simulación.*

2. Luego de finalizar la creación del proyecto, en la pantalla principal de Quartus II, seleccionar **Tools > Qsys**.

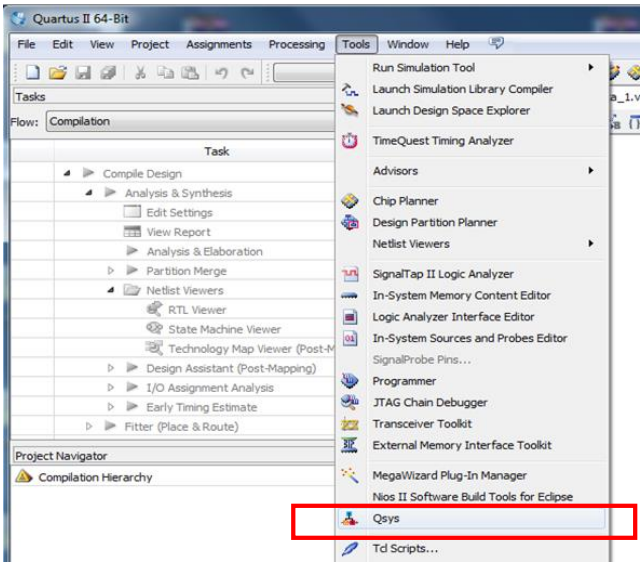


Figura 2. Selección de la herramienta Qsys

3. En la ventana de Qsys, en la parte izquierda se encuentra la librería de componentes (*Component Library*), añadir el procesador del sistema seleccionando la opción de **Embedded Processors** > **Nios II Processors** y dar doble clic (o en su defecto **Add**).

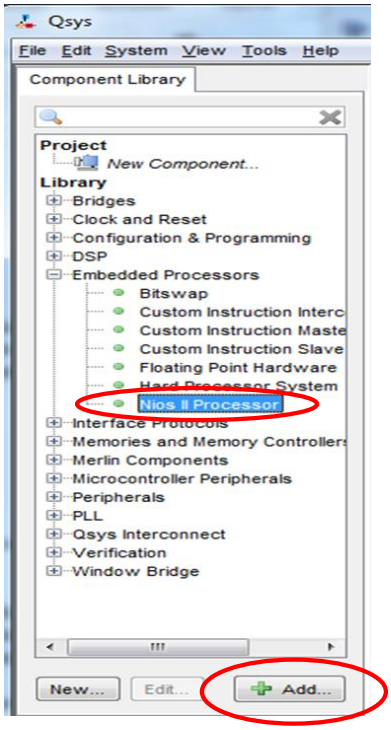


Figura 3. Adición del componente Nios II Processor al sistema de hardware

4. En la ventana que se abre, la primera pestaña permite seleccionar entre 3 opciones de núcleos de Nios II, las cuales son:

   Seleccionar el núcleo **Nios II/s**, que pese a superar las necesidades de esta práctica nos permite tener un balance entre rendimiento y utilización de recursos. Dejar las otras opciones por defecto y dar clic en **Finish**. En la pantalla principal de Qsys, en la pestaña "*System Contents*" dar clic derecho sobre el componente nyos2_qsys_0, seleccionar **Rename** y escribir cpu.

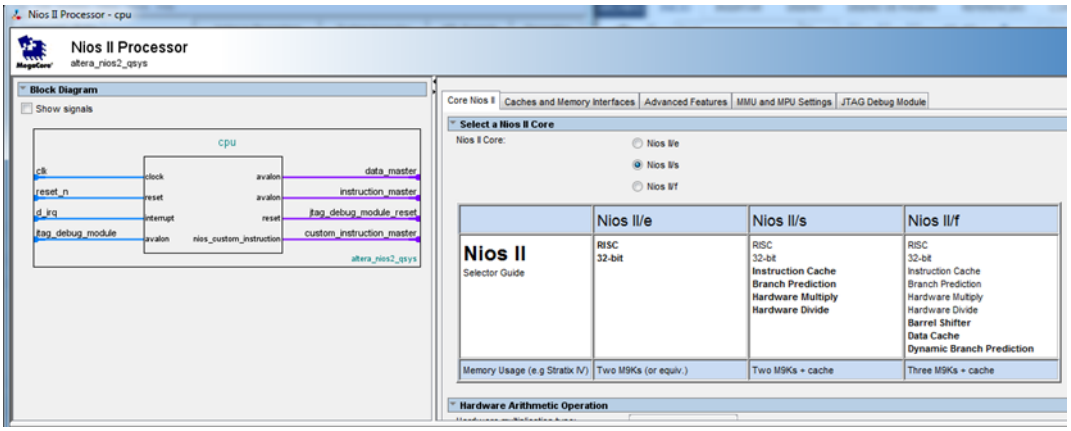


Figura 4. Selección del tipo de núcleo del procesador Nios II

En la parte inferior de la ventana de Qsys observamos mensajes de error (*véase Figura 5*), por ahora haremos caso omiso a los mismos debido a que se dan porque no se han realizado las conexiones correspondientes y asignación de direcciones a los componentes añadidos.

Figura 5. Mensajes de error por falta de configuraciones del sistema

5. Todo sistema debe contener al menos una memoria, para ello nos ubicamos en la librería de componentes y seleccionamos **Memories and Memory Controllers** > **On-Chip** > **On-Chip Memory (RAM or ROM)**, y dar clic en **Add**.

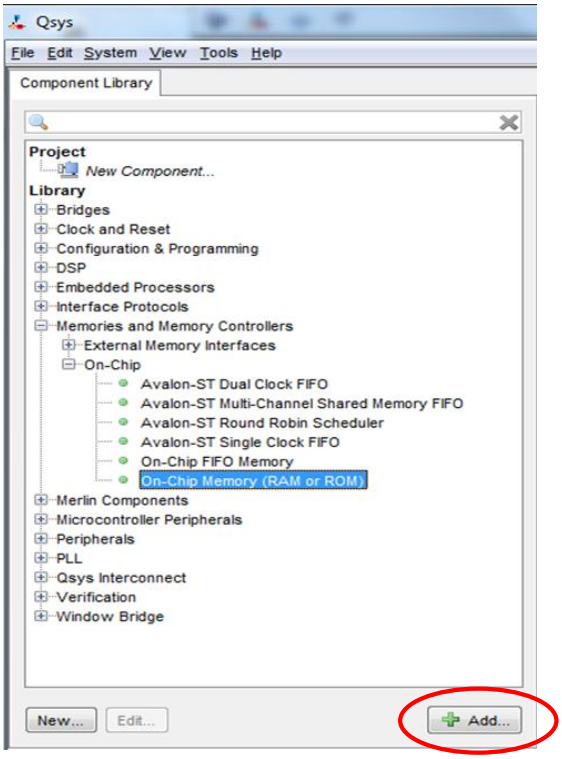


Figura 6. Adición de la memoria del sistema de hardware

Ajustar el tamaño de la memoria:

- Data width = 32

- Total memory size = 20480 bytes (equivalente a 20 KB)

Clic en **Finish**. Cambiar el nombre de onchip_memory2_0 por onchip_mem.

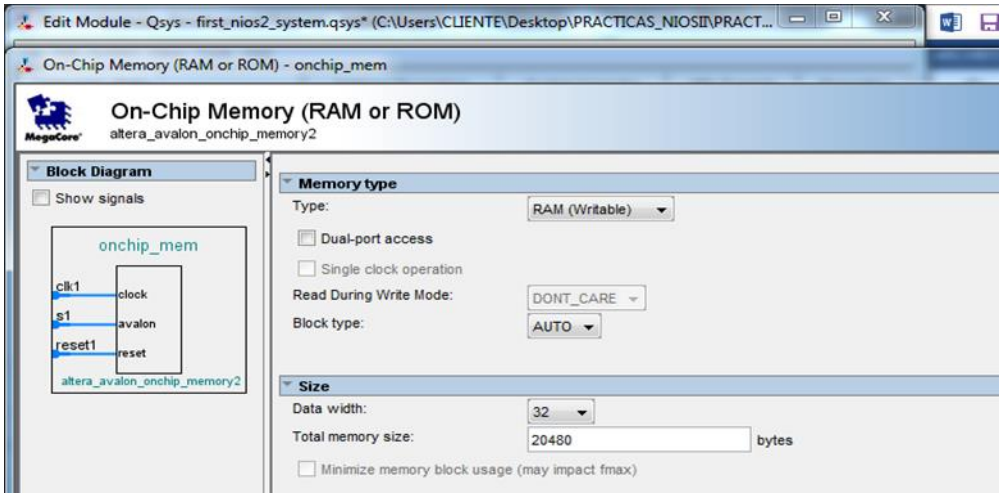


Figura 7. Configuraciones en los parámetros de la memoria del sistema.

6. Realizar las siguientes conexiones entre los elementos con los que contamos: clk_0, cpu y onchip_mem:
   - La entrada de reloj del procesador y de la memoria con el reloj de salida del componente clk_0.
   - Las entradas de Reset del procesador y de la memoria con el Reset de salida del componente clk_0 y con la salida *jtag_debug_module_reset* del procesador.
   - La entrada s1 de la memoria a las salidas *data_master* e *instruction_master* del procesador.

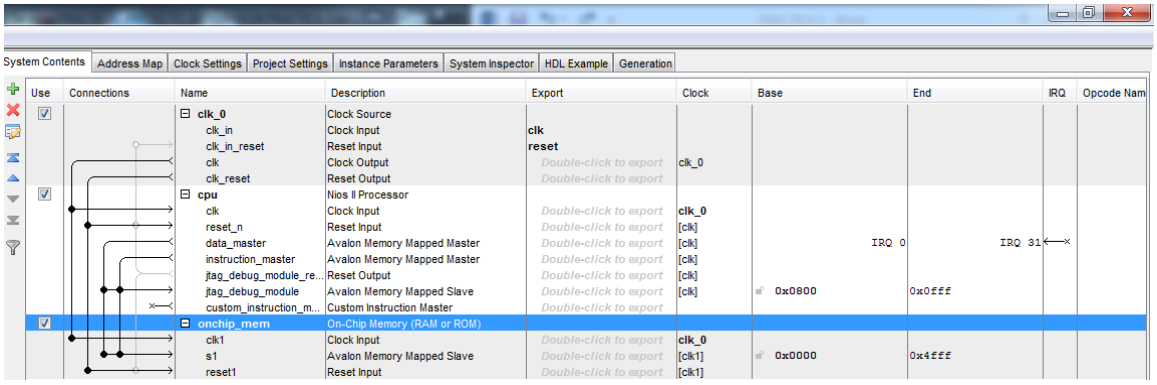


Figura 8. Conexiones entre el reloj, el procesador y la memoria del sistema

7. Abrir el editor de parámetros del Nios II processor (doble clic sobre cpu). En **Reset vector memory** seleccionar onchip_mem.s1 y escribir 0x00000000 en **Reset vector offset**. En **Exception vector memory** selccionar onchip_mem.s1 y escribir 0x00000020 en **Exception vector offset**.

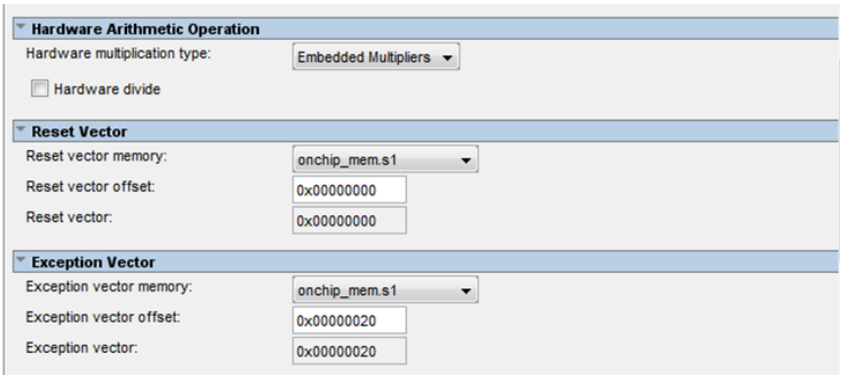


Figura 9. Ajuste del vector de reset y vector de excepción del sistema.

8. En la misma ventana del componente cpu, clic en la pestaña **Caches and Memory Interfaces** y establecer los ajustes como se muestran en la *Figura 10.* Luego de ello clic en **Finish**.

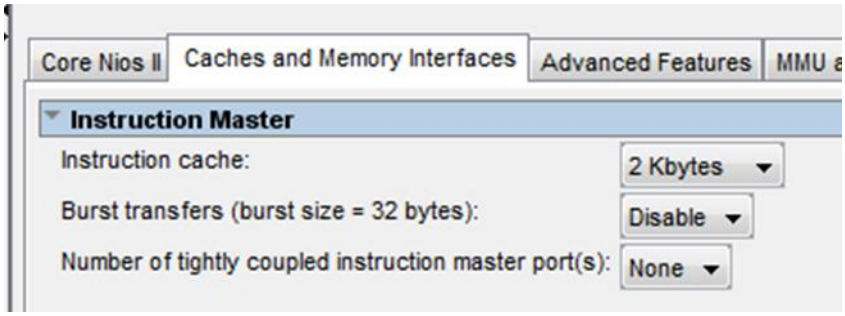


Figura 10. Ajuste de la interfaz de memoria y memoria cache del procesador.

9. Al regresar a la ventana de Qsys, seleccionar en la sección de librería de componentes la opción **Interface Protocols** > **Serial** > **JTAG UART**. Dejar los ajustes que están por defecto, clic en **Finish**. Renombrar el componente a jtag_uart.

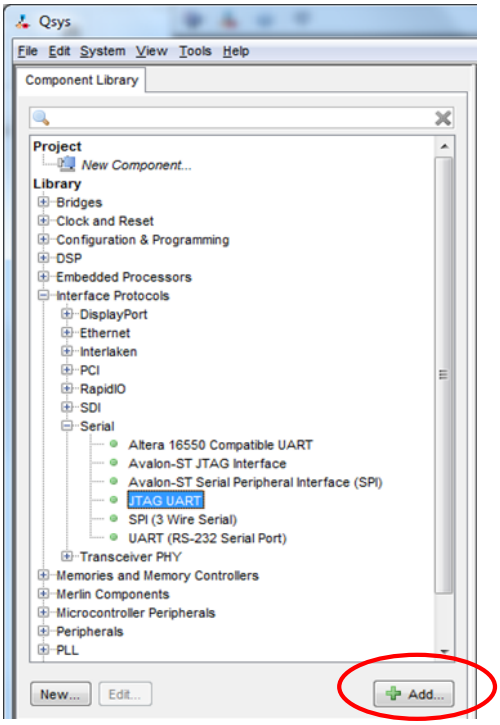


Figura 11.Adición del componente JTAG UART

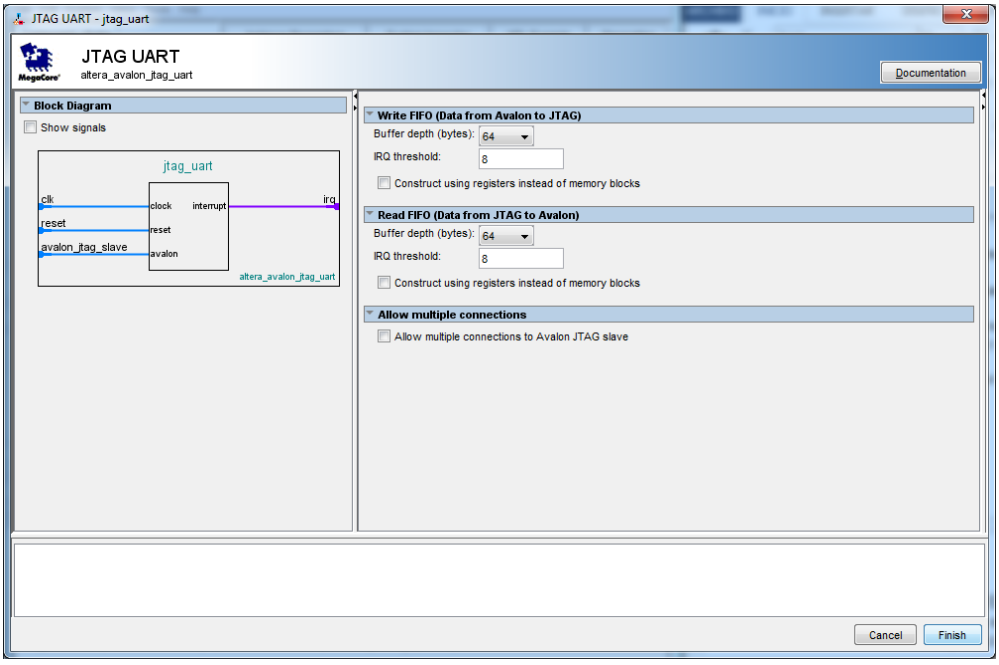


Figura 12.Ventana de ajustes de parámetros del componente JTAG UART

El JTAG UART provee un método conveniente de comunicar la computadora con el procesador Nios II a través del cable USB-Blaster.

10. Conectar los puertos *clk* y *clk_reset* del componente clk_0 a los puertos *clk* y *reset* del JTAG UART, respectivamente. Conectar el puerto *data_master* del procesador Nios II con el puerto *avalon_jtag_slave* del JTAG UART (*véase Figura 13*).

    Cabe señalar que el puerto *instruction_master* del procesador Nios II no se conecta al JTAG UART porque este último no es un dispositivo de memoria y por lo tanto no puede enviar instrucciones al componente cpu.

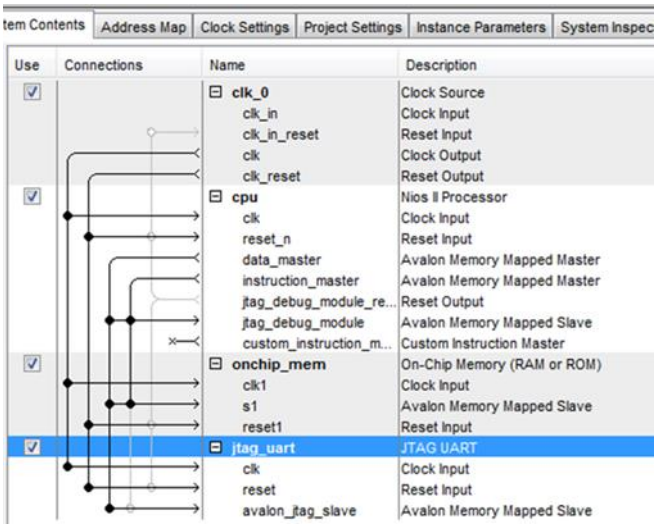


Figura 13. Conexión del componente JTAG UART con el resto del sistema

11. La mayoría de los sistemas de control utilizan un temporizador para habilitar la precisión en el cálculo del tiempo. Seleccionar en la sección de librería de componentes la opción **Peripherials** > **Microcontroller Peripherials** > **Interval Timer**. No variar los ajustes que se encuentran por defecto. Cambiar el nombre del componente a **sys_clk_timer**.

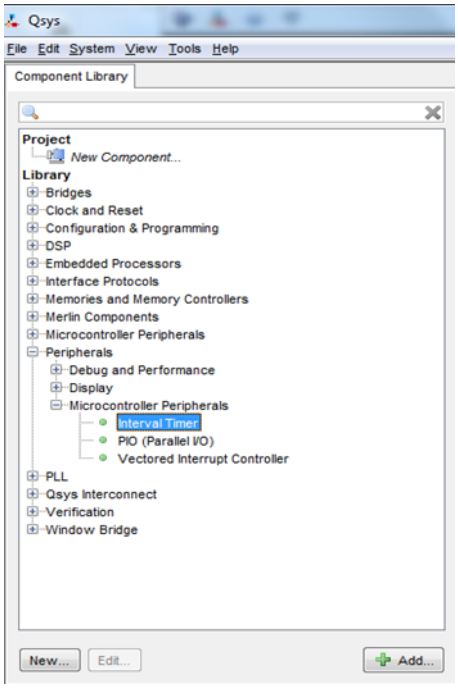


Figura 14. Adición de un componente temporizador al sistema

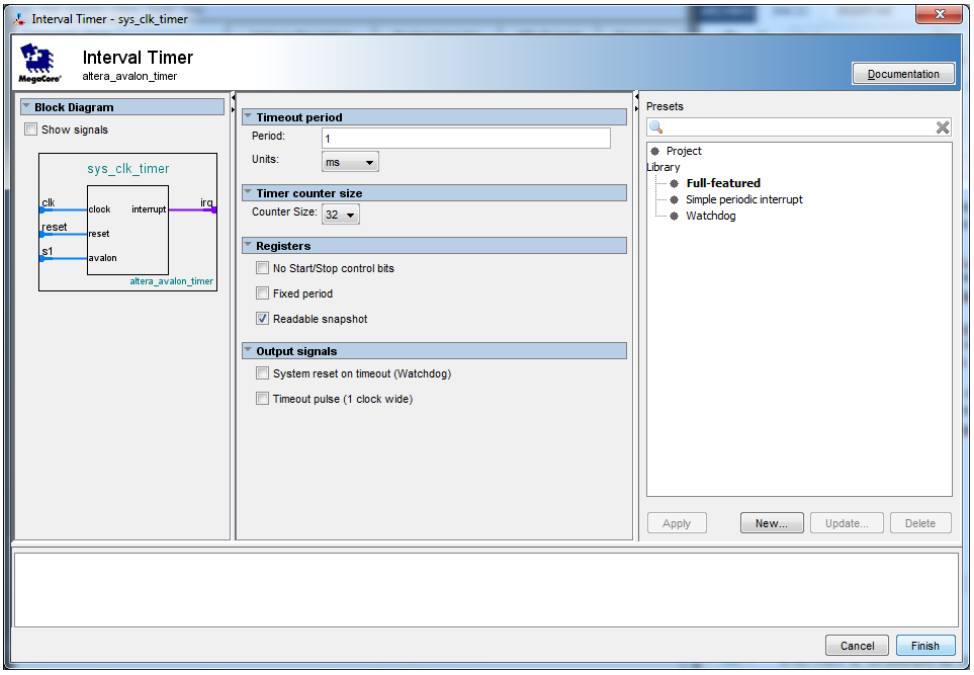


Figura 15.Ventana de ajustes de parámetros del componente Interval Timer

12. Conectar los puertos *clk* y *clk_reset* del componente clk_0 a los puertos *clk* y *reset* del Interval Timer, respectivamente. Conectar el puerto *data_master* del procesador Nios II con el puerto *s1* del Interval Timer (*véase Figura 18*).

13. Añadir de la sección librería de componentes la opción **Peripherials** > **Debug and Performance** > **System ID Peripherial**. Cambiar el nombre del componente a sysid.

    Este componente se añade para evitar accidentalmente descargar algún otro programa compilado para otro sistema.

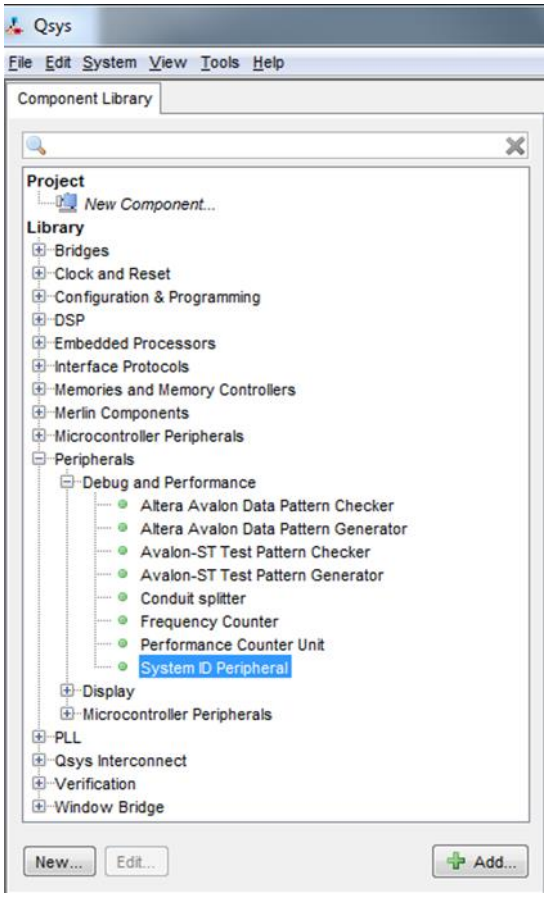


Figura 16. Adición del componente System ID Peripherial

14. Conectar los puertos clk y clk_reset del componente clk_0 a los puertos clk y reset del System ID peripherial, respectivamente. Conectar el puerto

data_master del procesador Nios II con el puerto *control_slave* del System ID peripherial (*véase Figura 18*).

15. Para dirigir las señales de salida se utiliza el módulo PIO. Añadir de la sección librería de componentes la opción **Peripherials** > **Microcontroller Peripherials** > **PIO (Parallel I/O)**. Cambiar el nombre del componente a leds. Realizar los siguientes ajustes en la ventana de parámetros de este componente:
    - Width (1-32bits): 8
    - Direction: Seleccionar Output

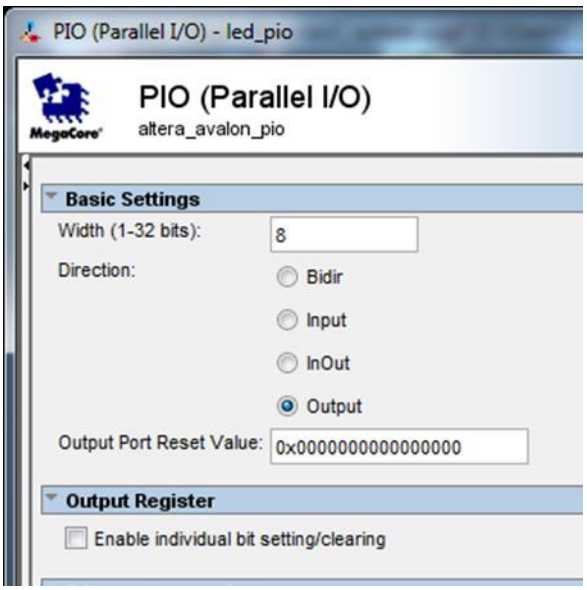


Figura 17. Configuraciones en los parámetros del componente PIO

16. Conectar los puertos clk y clk_reset del componente clk_0 a los puertos clk y reset del PIO, respectivamente. Conectar el puerto *data_master* del procesador Nios II con el puerto *s1* del PIO (*véase Figura 18)*.

17. En la fila de external_connection, doble clic en **Click to export** en la columna **Export** para exportar así los puertos PIO (*véase Figura 18)*.

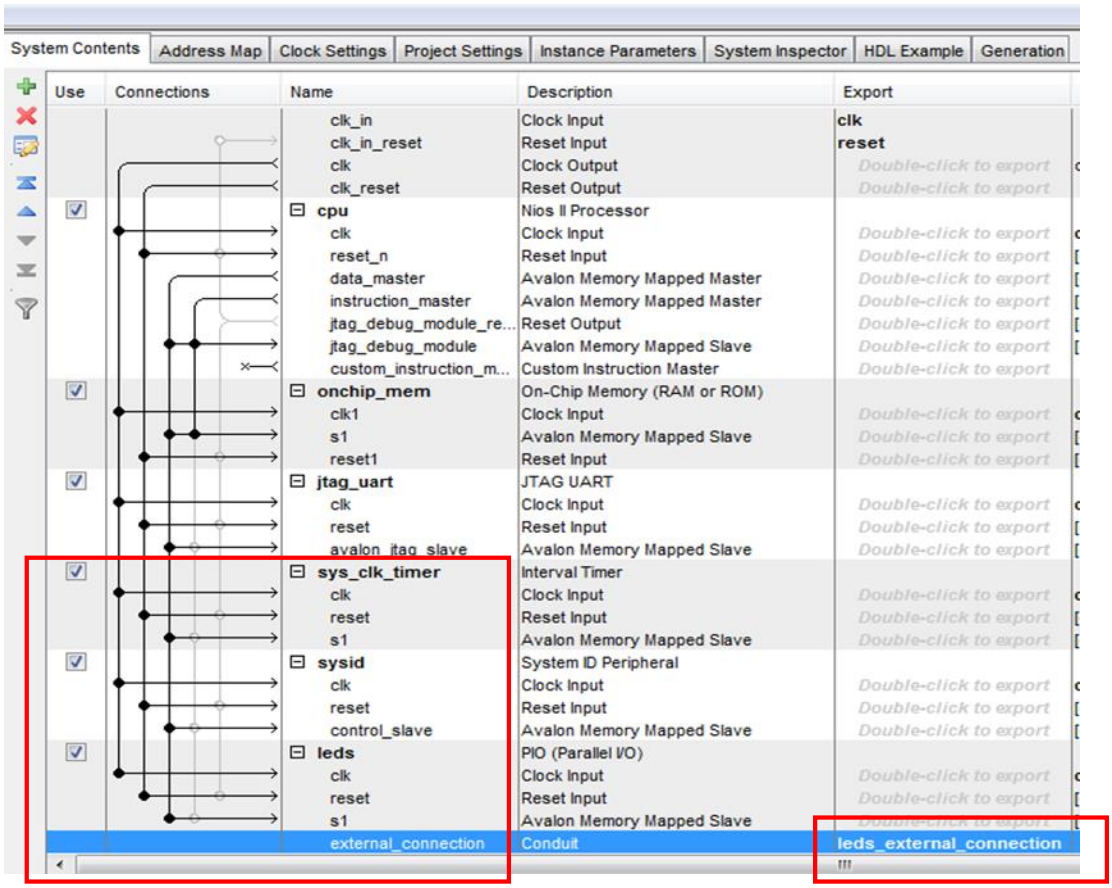


Figura 18. Conexiones de los componentes del sistema de hardware y señales exportadas.

18. Una vez que se ha añadido todos los componentes del hardware que se requiere, se debe especificar cómo interactúan para formar el sistema, para ello: seleccionar en la ventana principal de Qsys la pestaña **System** > **Assign Base Addresses**, con lo cual se asigna automáticamente la dirección base a cada componente. Se observa que los errores se han terminado de borrar.

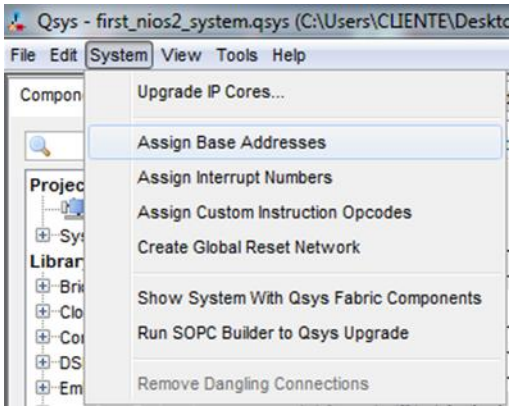


Figura 19. Asignación automática de direcciones de memoria a los componentes del sistema de hardware Nios II.

19. Se debe añadir las solicitudes de interrupción o IRQs (Interrupt Request), esto para producir resultados válidos de hardware. El componente Interval Timer debe tener el IRQ de mayor prioridad para mantener la exactitud del sistema. En la columna IRQ, conectar el Procesador Nios II al JTAG UART y al Interval Timer. Clic en el valor IRQ del jtag_uart y escribir 16, de manera similar escribir 1 para el valor IRQ del componente sys_clk_timer.

    Un IRQ es una señal de un sistema de hardware que indica que el procesador haga algo, pero en base a su prioridad se atiende la solicitud. Los valores bajos de IRQs son considerados de mayor prioridad.

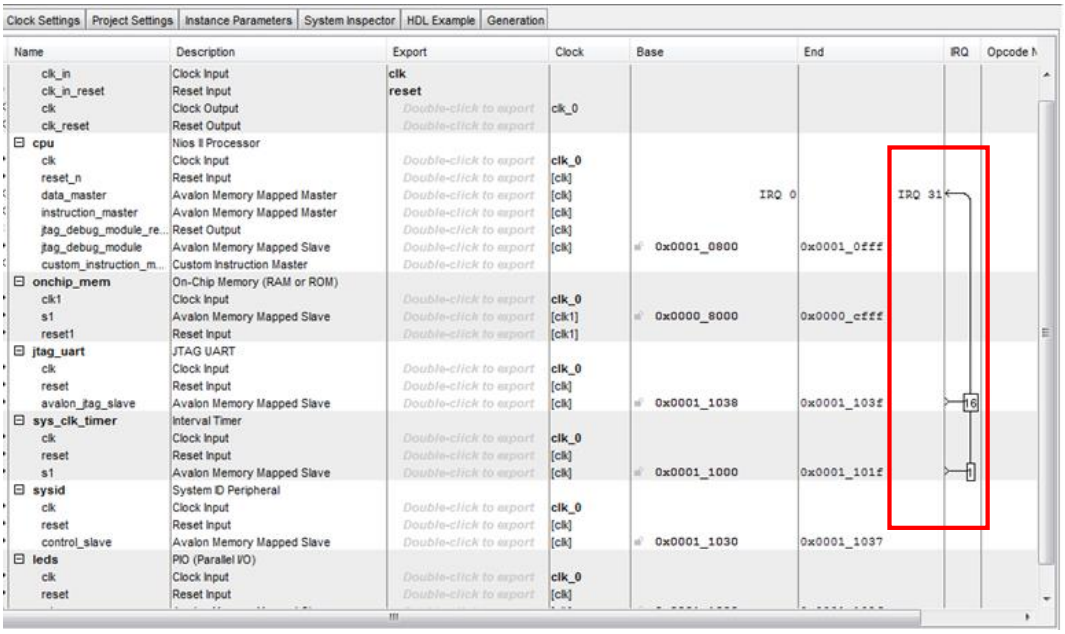


Figura 20. Adición de la prioridad en la solicitud de interrupciones del sistema.

20. Guardar, llenar el nombre de archivo (*sugerido first_system*), clic en guardar.
21. En la ventana principal de Qsys, dar clic en la pestaña **Generation** y seleccionar None tanto en la sección **Simulation** como en **Testbench System.** Clic **Generate,** al finalizar clic en **Close,** cerrar Qsys y regresar a Quartus II.

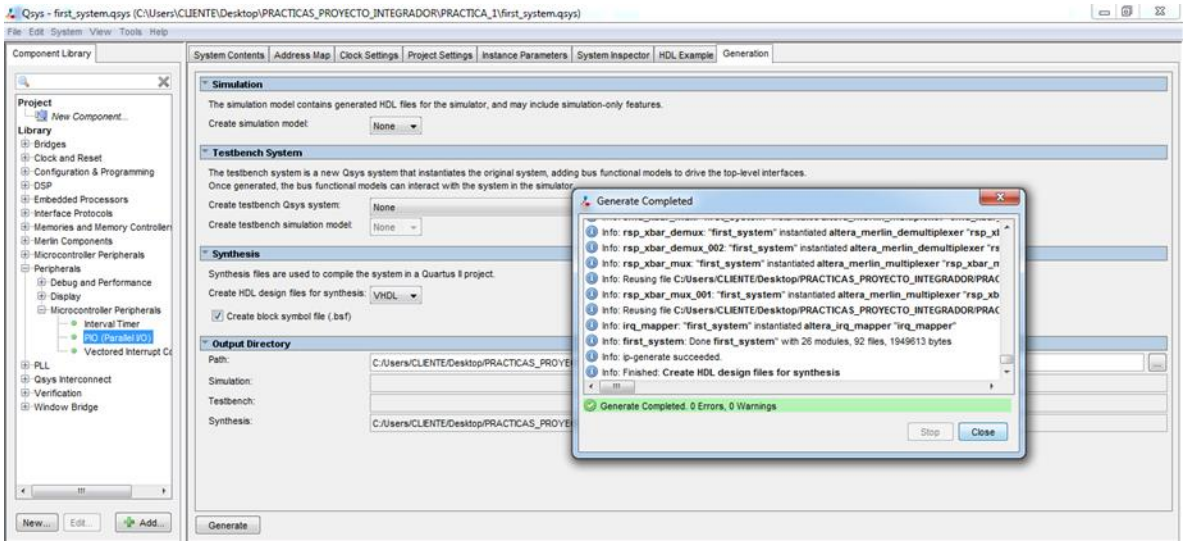


Figura 21. Generación del sistema en Qsys.

22. Se procede a instanciar el módulo del sistema creado en Qsys, para ello crear un nuevo *Block Diagram / Schematic File*, y añadir el bloque first_system. Agregar los terminales de entrada y salida, con sus respectivas etiquetas. Guardar el archivo (practica_1).

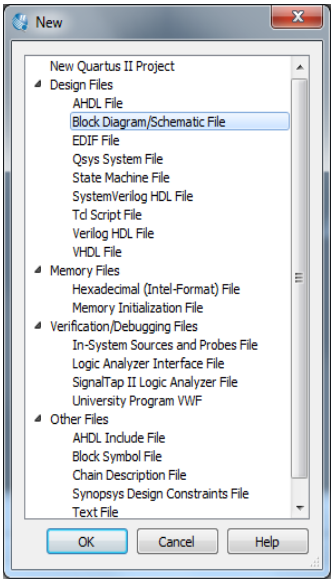


Figura 21. Creación de un archivo de Diagrama de Bloques en Quartus II.

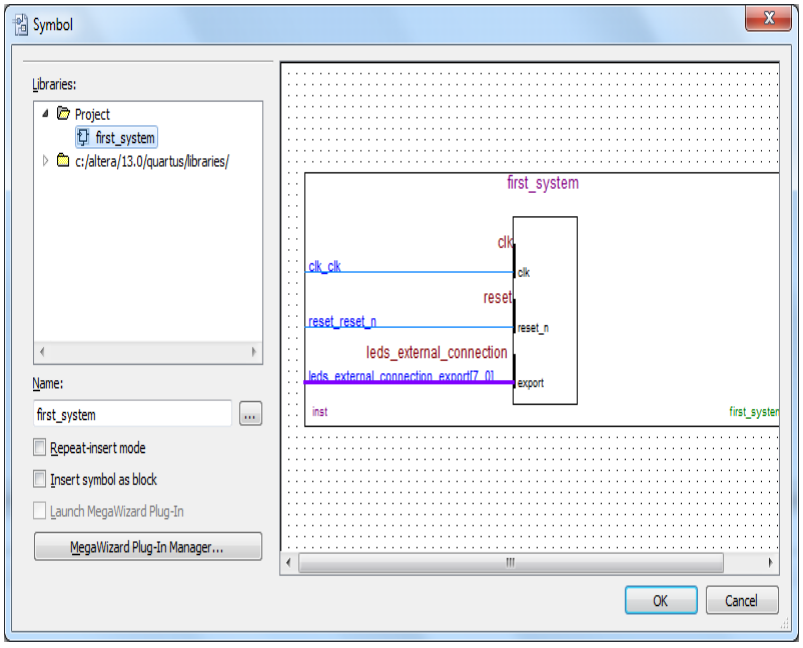


Figura 22. Adición del sistema de hardware creado en Qsys en el diagrama de bloques.

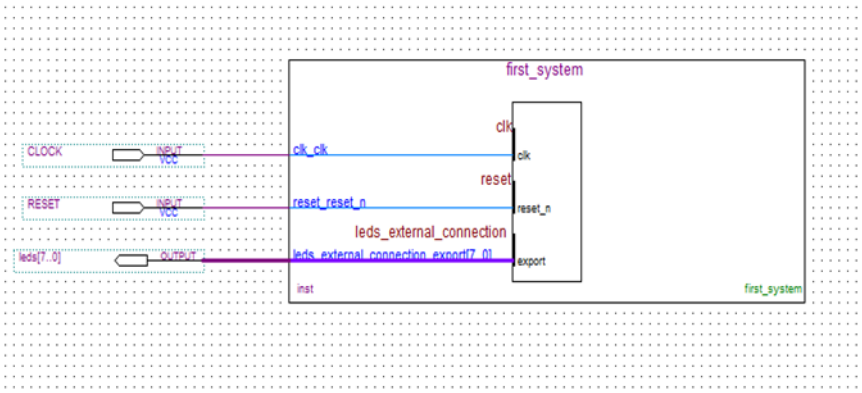


Figura 23. Adición de los pines al nuevo bloque first_system

23. Se procede a añadir el archivo de extensión .qip generado en Qsys, que contiene la información del sistema Nios II creado. En la ventana de **Project Navigator**, dar doble clic sobre la carpeta **Files**, y buscar el archivo que se encuentra en la ruta: *<nombre del directorio>/first_system_synthesis/first_system.qip* (*Verificar que se esté mostrando todos los archivos o los de extensión .qip al buscar*)
24. Para realizar la asignación de pines: en el menú **Processing** seleccionar **Start** > **Start Analysis & Elaboration,** con lo cual se genera el archivo .qsf que contiene información del proyecto. Al revisar el **Pin Planner (Assigments > Pin Planner)** observamos las entradas y salidas creadas.

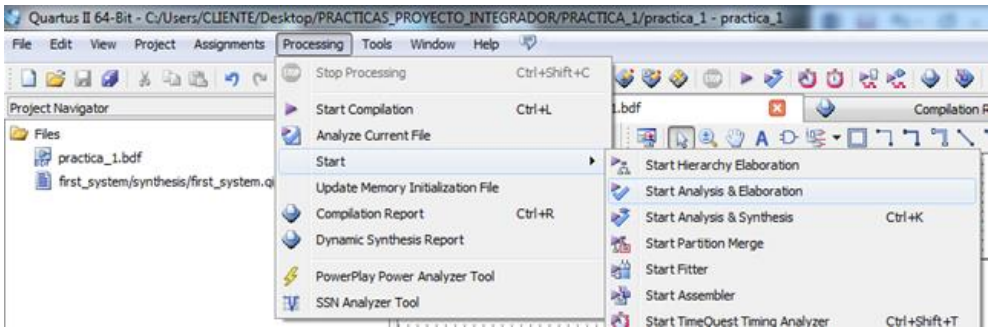


Figura 24. Generación del archivo .qsf para poder realizar la asignación de pines

25. Realizar la asignación de pines como sigue:
    - CLOCK > PIN_R8
    - RESET> PIN_J15 (Corresponde al pulsador KEY0 de la tarjeta DE0-Nano)
    - leds[0] >PIN_A15
    - leds [1] >PIN_A13
    - leds [2] >PIN_B13
    - leds [3] >PIN_A11
    - leds [4] >PIN_D1
    - leds [5] >PIN_F3
    - leds [6] >PIN_B1
    - leds [7] >PIN_L3

    Los pines asignados para los leds corresponden a los 8 LEDs de la tarjeta DE0-Nano. (*Información que se extrae del manual de usuario de la tarjeta*).

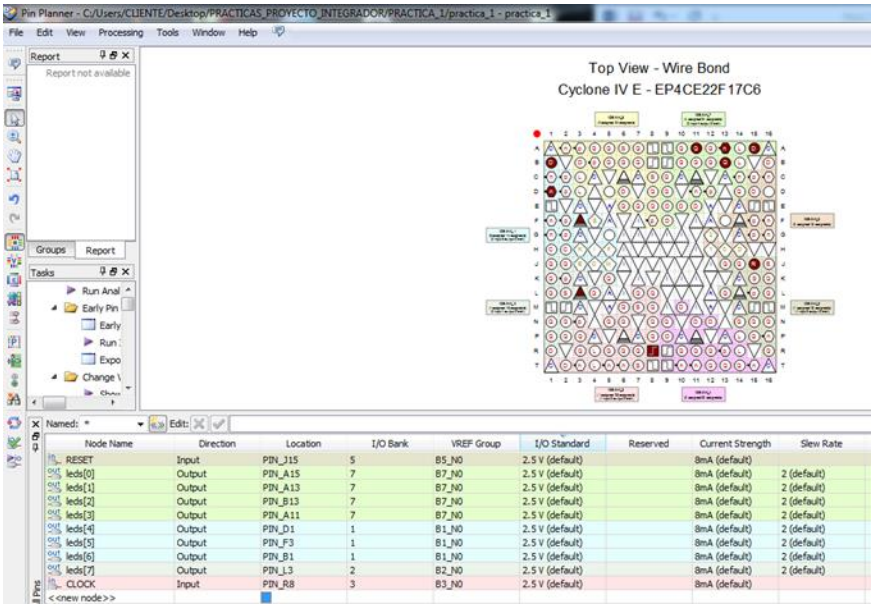


Figura 25. Ventana del Pin Planner

26. Compilar el proyecto para crear el archivo .sof que se descargará en la tarjeta, luego de lo cual podemos observar el reporte de compilación que fue exitoso y la utilización de elementos lógicos de la FPGA.

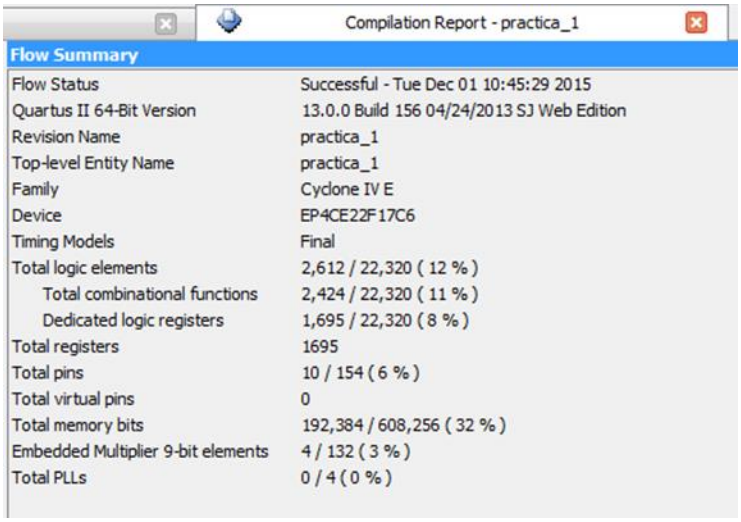


Figura 26. Reporte al finalizar la compilación en Quartus II

27. Conectar la tarjeta DE0- Nano a la computadora mediante el cable USB-Blaster. Se abre la herramienta de programación, seleccionar **Tools > Programmer.**

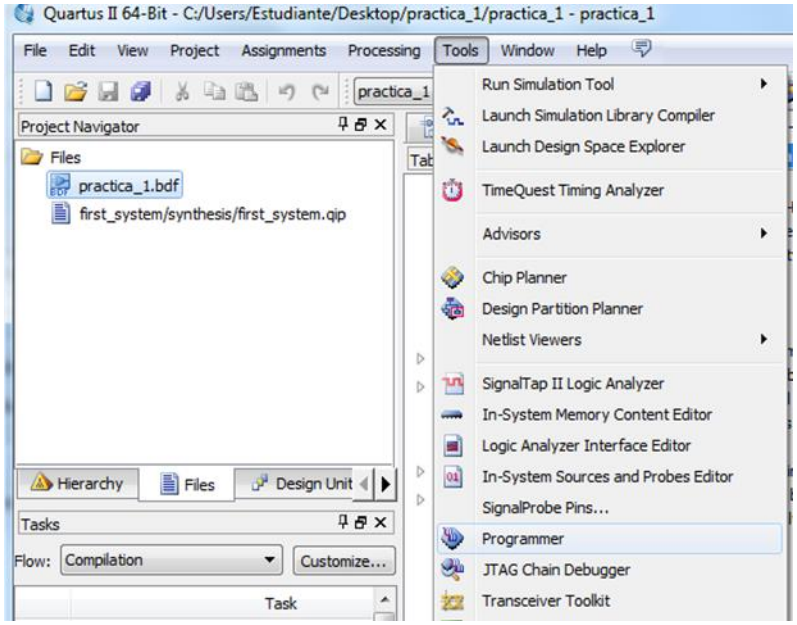


Figura 27. Selección de la herramienta de programación de Quartus II

28. En la ventana del programador, verificar que se haya reconocido el USB-Blaster, caso contrario clic sobre **Hardware Setup**, buscarlo, seleccionarlo y aceptar las configuraciones. En la ventana principal del programador clic en **Add File** y buscar el archivo de extensión .sof que se generó en la compilación, en la ruta <carpeta del directorio>/output_files/practica_1.sof. Luego de añadirlo, clic en **Start**.

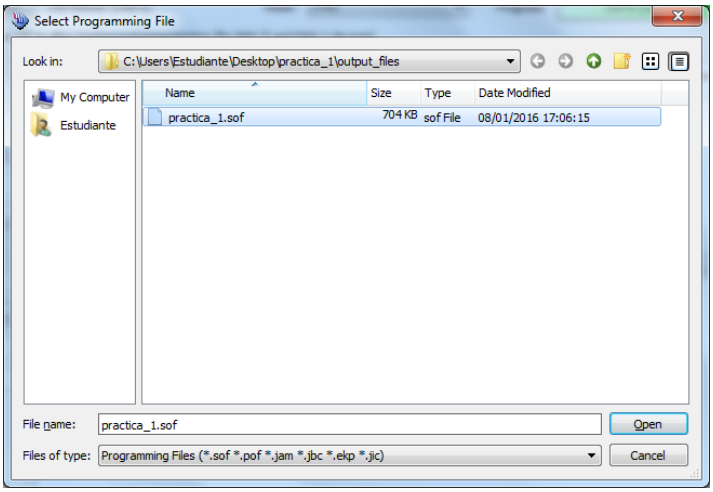


Figura 28 Búsqueda del archivo .sof que se guardará en la memoria volátil de la FPGA.

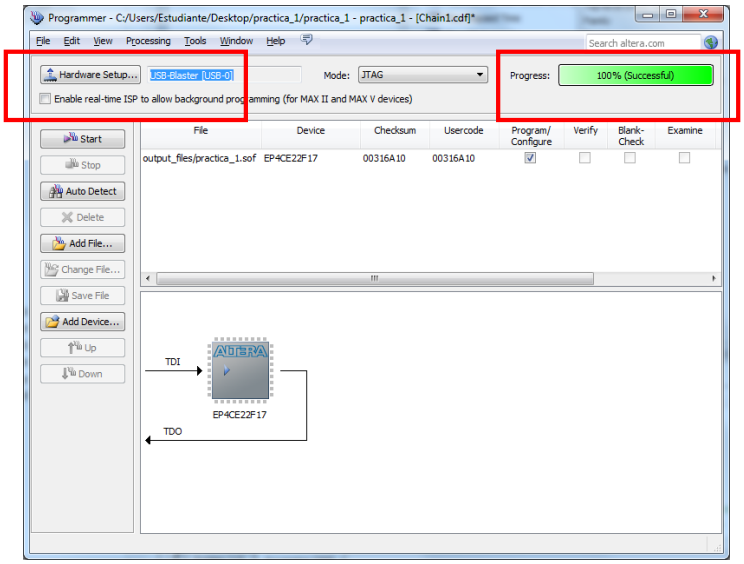


Figura 29. Luego de dar clic en Start, la compilación debe ser exitosa

El paso 29 es de interés para quien trabaje con Quartus II Web Edition y utilice el procesador Nios ii en su versión /s o /f

29. Podría darse el caso en que el archivo generado luego de la compilación sea de tipo time_limited.sof. Si este es el caso se mostraría una notificación al abrir el programador que se debe al uso del procesador nios ii en su versión /s o /f, teniendo Quartus II Web Edition (*conocido en la versión 15.1 como Quartus Prime Lite Edition*), en caso de utilizar Nios II /e o la versión estándar de Quartus II no se da esta advertencia y se obtiene un archivo .sof luego de la compilación; pero igual se puede trabajar con el time_limited.sof, por lo tanto daría clic en **OK.**

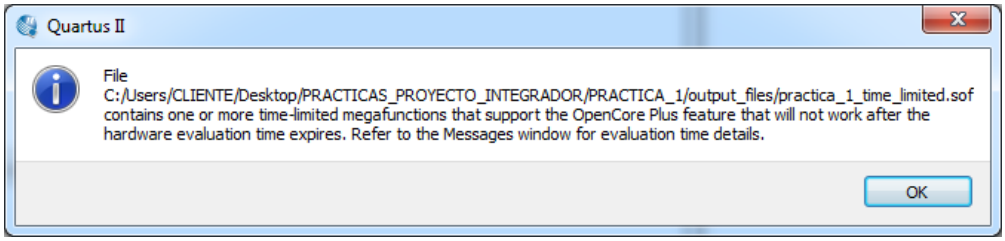


Figura 30. Ventana al abrir el programador en Quartus II Web Edition y utilizar el procesador Nios ii en su versión /s o /f

En la ventana de Programmer se da clic en Start, luego de finalizar de cargar el programa en la tarjeta sale una ventana que no debe cerrarse mientras se está trabajando, esto se debe a lo que se mencionaba anteriormente en cuanto al uso de Nios II /s como procesador.

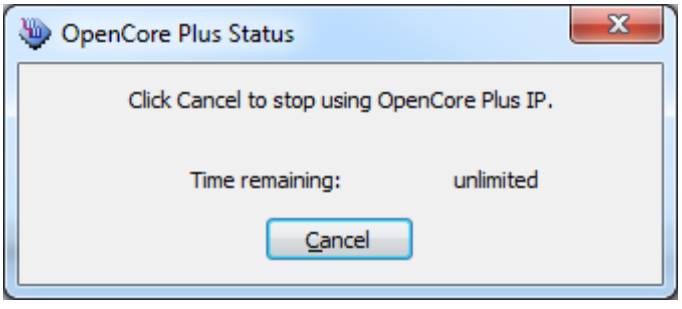


Figura 31. Ventana al finalizar la programación de la DE0-Nano en Quartus II Web Edition y utilizar el procesador Nios ii en su versión /s o /f. (No se debe cerrar)

30. En la ventana de Quartus II, en el menú Tools, seleccionar **Nios II Software Build Tools for Eclipse.**

    Con esta herramienta compilaremos un programa básico en lenguaje C para ejecutarlo sobre el sistema de hardware creado anteriormente.

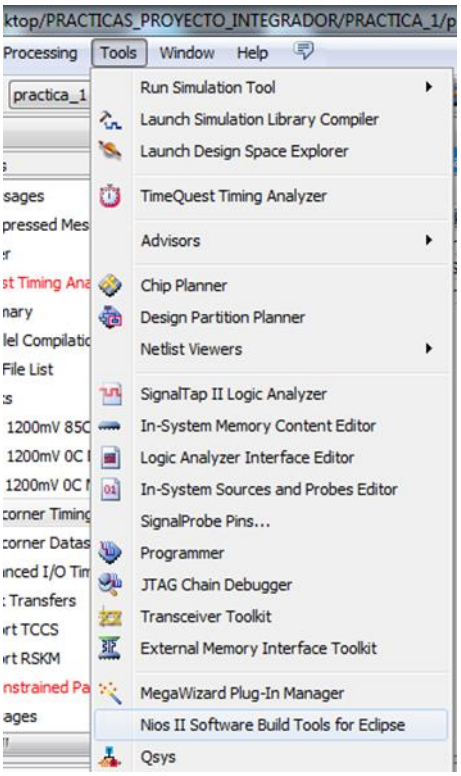


Figura 32. Selección de la herramienta Nios II SBT for Eclipse

Cuando se abre la ventana Workspace Launcher, crear una carpeta en el directorio del proyecto con el nombre workspace, colocar la dirección de la carpeta creada y click en **OK.**

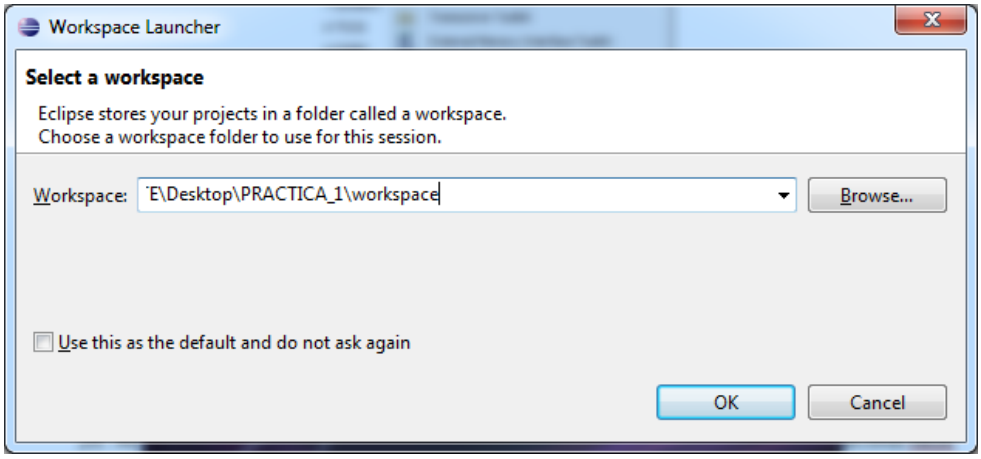


Figura 33. Selección del espacio de trabajo para la creación del software

31. Antes de crear un proyecto de Nios II debemos asegurarnos que la perspectiva Nios II este visible. Para ello se sigue las ruta desde el menú **Window > Open Perspective > Other.**

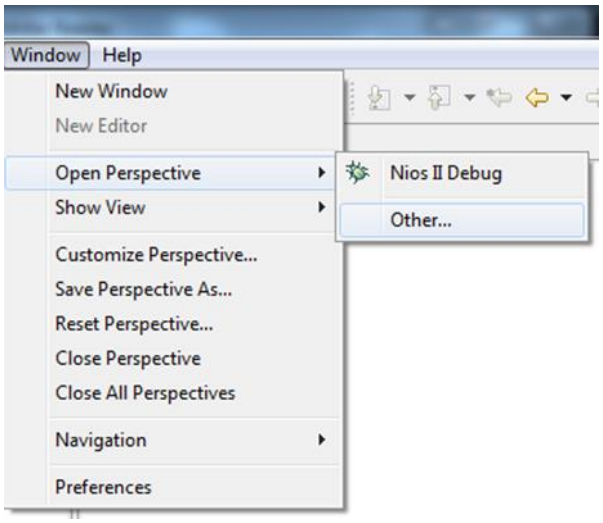


Figura 34. Ruta para el cambio de perspectiva en Nios II SBT

Una vez que damos clic en Other se nos abre una ventana en la cual seleccionamos Nios II y clic en **OK.** Esto nos permite que el entorno presente las ventanas necesarias para nuestro trabajo de desarrollar un programa sobre el procesador Nios II y no nos muestre ventanas innecesarias para este propósito.

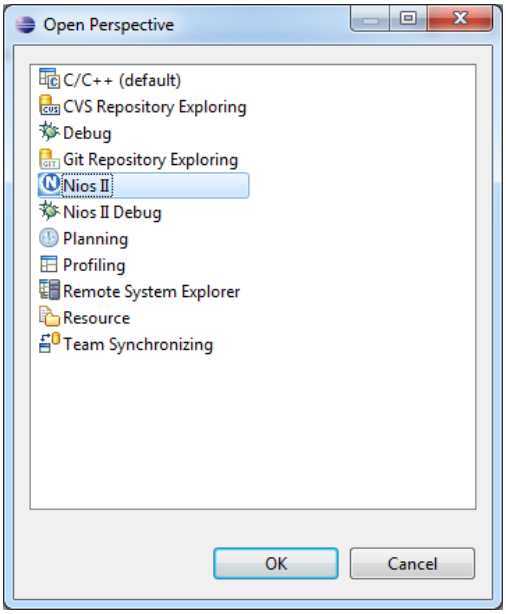


Figura 35. Selección de la perspectiva Nios II, para adecuar el entorno.

32. Utilizaremos una plantilla del programa. Para ello, **File** > **New** > **Nios II Application and BSP from Template.**

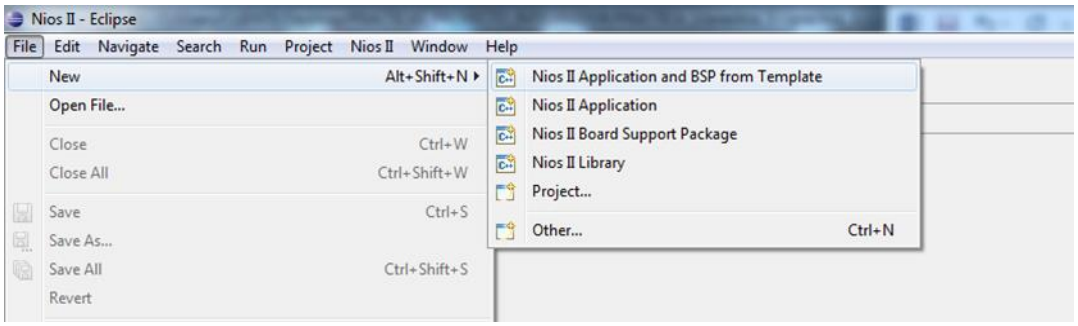


Figura 36. Búsqueda de las plantilas de Nios II SBT

33. En SOPC Information File Name se busca el archivo first_system.sopcinfo, y automáticamente se llena la opción CPU name, con el nombre que se le dio al procesador, cpu (*Véase Figura 37*). Añadir un nombre al proyecto (sugerido: contador) y se selecciona una plantilla, en Templates escoger Blank Project. Clic en **Finish**.

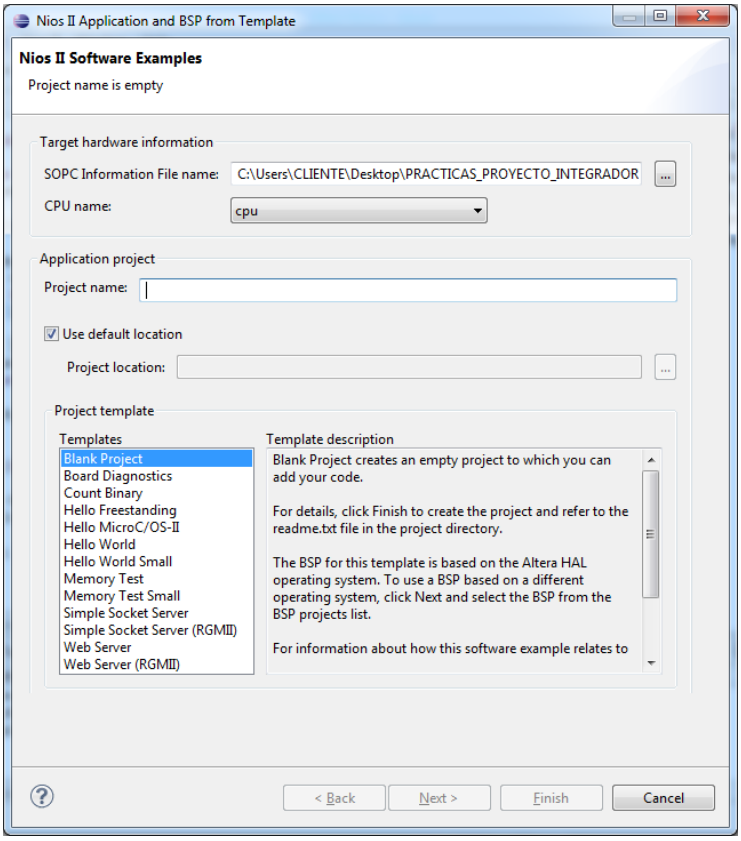


Figura 37. Creación del proyecto de software en base al archivo .sopcinfo

En la ventana Project Explorer se observan los siguientes nuevos proyectos:

- contador.- Contiene la aplicación del proyecto C/C++
- contador_binario.bsp.- Un paquete de sorporte de la tarjeta que encapsula todos los detalles del sistema de hardware Nios II

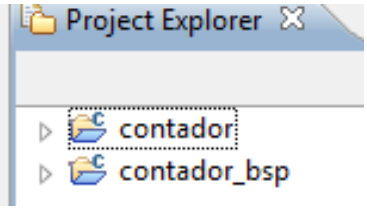


Figura 38. Proyecto de aplicación (contador) y proyecto de soporte específico del sistema (contador_bsp)

34. Clic derecho sobre el proyecto *contador_bsp* y seleccionar **Properties.** En la opción *Nios II BSP Properties* se habilitan únicamente las opciones *Reduced device drivers* y *Small C library,* las demás no se seleccionan.

    Esto se realiza para minimizar el consumo de memoria del software debido a que nuestro sistema tiene sólo 20 KB de memoria y parte de la misma como se observa en el reporte de compilación se usó para guardar el sistema de hardware.

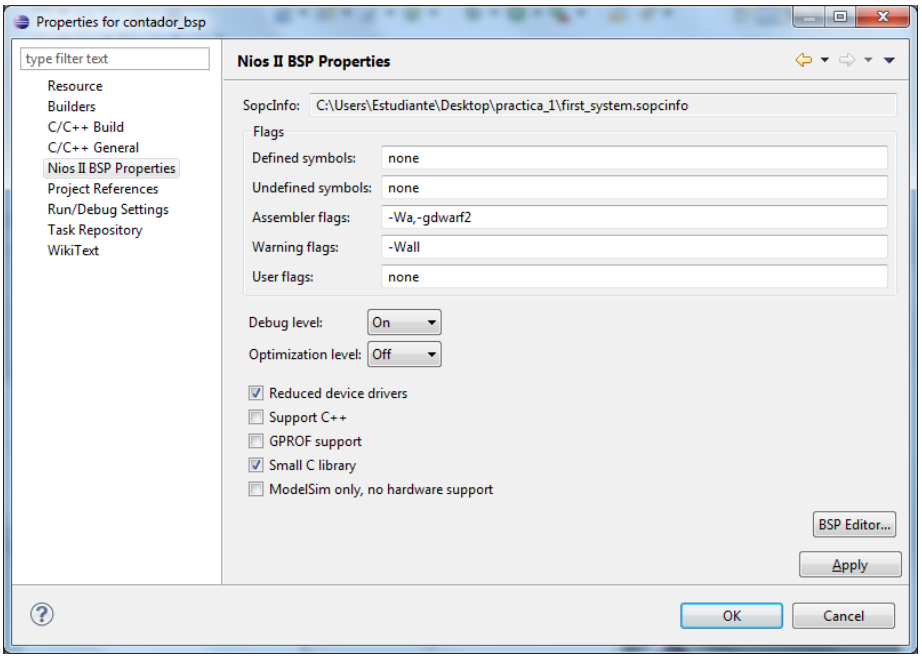


Figura 39. Propiedades del proyecto de soporte (contador_bsp)

35. Clic derecho sobre el proyecto contador y seleccionar **New > File**. En el nombre del archivo escriba *main.c* (Es importante poner la extensión .c para que lo reconozca como código fuente) y clic en **Finish**.

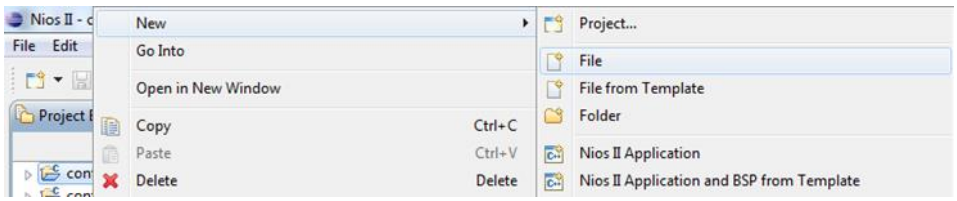


Figura 40. Creación de un archivo en el proyecto de aplicación (contador)

En el archivo creado copie el siguiente código:

```
#include <stdlib.h>
#include <stdio.h>

#define LED_ADDR 0x00011020 //espacio de memoria leds

int main(void)
{
		volatile int *led = (int*)(LED_ADDR); //asignacion del puntero
al espacio de memoria
		int count=0; //contador que carga el valor en el LED

		while (1)
		{
		Delay (1000000);
			count++;
			if (count==256)
			{
				count = 0;
			}
		*(led) = count;
		printf("El número es: %d \n ", count);
		}
		return 0;
}

void Delay (int _Tiempo)
{
		int i = 0;
		while (i<_Tiempo)
		{
		i++;
		}
}
```

Nótese que la dirección de memoria asignada a LED_ADDR a la cual se apunta para escribir corresponde a la dirección base generada por Qsys para el componente PIO de los leds de la tarjeta DE0-Nano.

36. Clic derecho sobre el proyecto *contador* y seleccionar **Build Project**, al finalizar se muestra un mensaje de finalizado. Nuevamente, clic derecho sobre el proyecto *contador* y seleccionar **Run As** > **Nios II Hardware.**

    En la ventana de consola se visualiza la salida periódica del mensaje con el número que a su vez se visualizan en los leds de la tarjeta DE0-Nano cuyo patrón es el de un contador binario de 8 bits. Al llegar al final FF (255) reinicia su conteo, esto en base al código ingresado. Con el botón cuadrado rojo en la esquina superior derecha de la consola se puede detener la ejecución del programa.

Figura 41. Consola de Nios II SBT para la visualización de la aplicación

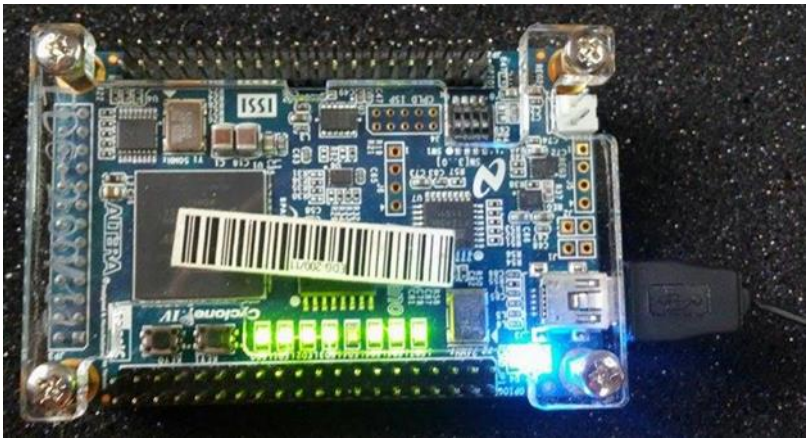


Figura 42. Visualización del contador incremental en los LEDS de la DE0-Nano

PRÁCTICA # 2

TEMA: *Lectura y escritura sobre el chip SDRAM integrado en la tarjeta DE0-Nano y control de una pantalla LCD 16x2*

1. Objetivos:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Entender y explicar el uso de memorias volátiles de mayor capacidad respecto a las on-chip RAM en el desarrollo de sistemas embebidos sobre el procesador Nios II.
- Emplear los componentes necesarios de Qsys para controlar la memoria SDRAM de la tarjeta DE0-Nano, tomando en cuenta *desplazamientos de reloj (clock skew)*
- Añadir los pines de control y datos en hardware para una pantalla LCD 16x2.
- Ejecutar una aplicación de software sobre el hardware desarrollado utilizando *'Nios II Software Build for Eclipse'.*

2. Requisitos mínimos:

- Tener instalado Quartus II y Nios II EDS (versión 13.0).
- Conocimientos básicos de Quartus II y programación en VHDL y C.
- Haber culminado satisfactoriamente la Práctica # 1.
- Lectura y comprensión del archivo: '*FUNDAMENTACIÓN_TEÓRICA_PRÁCTICA_SDRAM*

3. Breve explicación de la práctica:

Se desarrolla un sistema de hardware que incluye un procesador embebido de Altera Nios II en conjunto con otros módulos *(véase Figura 1)*, conectándose por medio de una interconexión llamada *Avalon switch fabric.* De esta manera se forma un sistema que luego se implementa en el chip FPGA utilizando el programa de *Quartus II*.

Esta práctica inicia desde el sistema de hardware desarrollado en la práctica 1 y se añade un controlador para la memoria volátil SDRAM contenida en la tarjeta DE0-Nano, otro componente con configuraciones específicas para satisfacer las necesidades de tiempo de la memoria añadida y se adiciona en hardware puertos de salidas para las señales de control y datos de una pantalla LCD 16x2.

Sobre el diseño de hardware desarrollado se corre dos programas creados *en Nios II Software Build Tools for Eclipse.* En el primero se realiza la escritura y lectura en el chip SDRAM, con visualización en la consola de Nios II SBT de un contador descendente de 8 bits, también se muestra la dinámica del contador en los leds de la tarjeta DE0-Nano. En el segundo programa se presenta el control por software de la LCD 16x2, mediante el acceso a los registros de memoria asignados en partes previas del flujo del diseño.

La importancia de esta práctica radica en que en aplicaciones comunes se requieren mayores cantidades de memoria para el programa y para ello se usan memorias externas como la SDRAM de la DE0-Nano. En las prácticas posteriores se utilizará este chip para evitar desbordes de memorias. Además el uso de pantallas LCD, es fundamental al momento de presentar datos al usuario, en la práctica se muestra también las consideraciones a tener en cuenta en caso de usar una LCD 20x4.

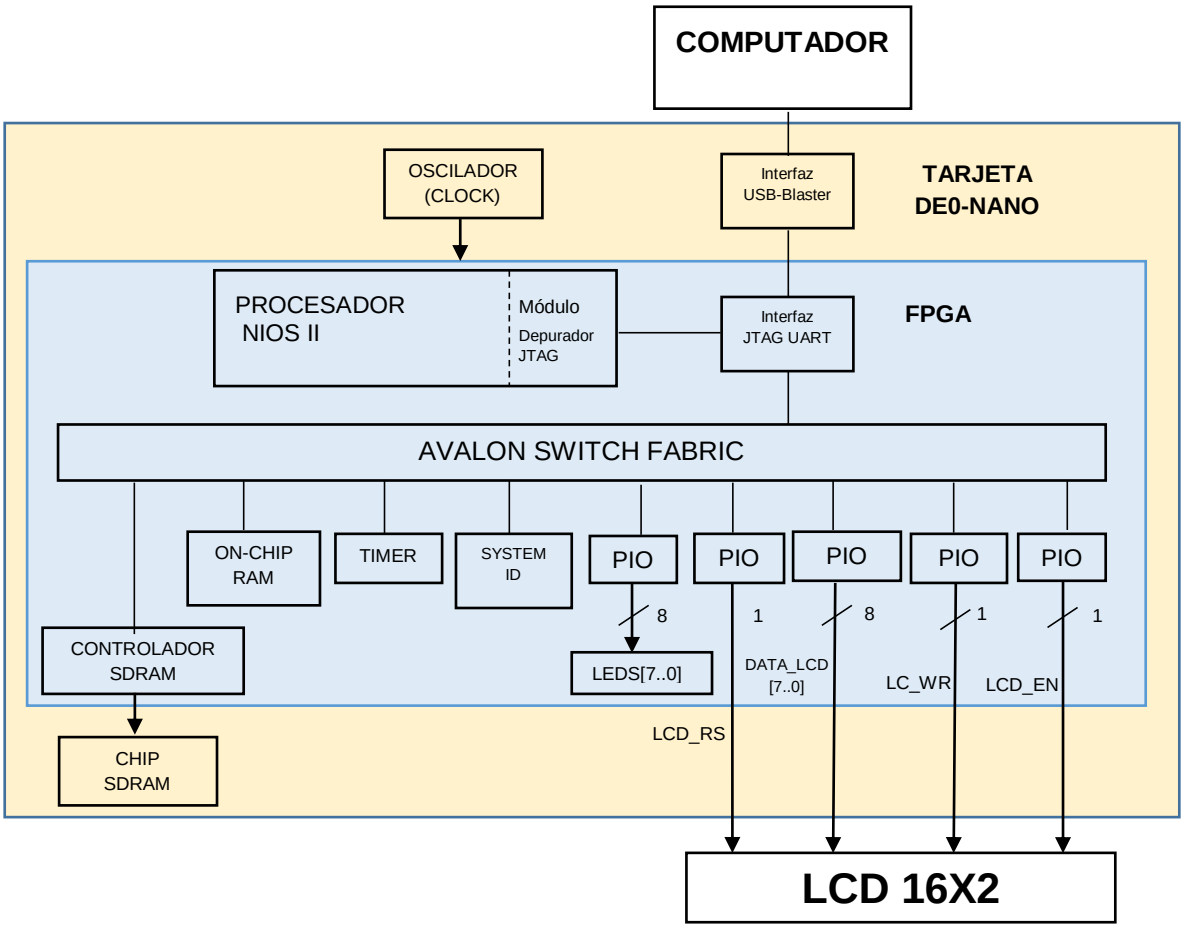


Figura 1. Diagrama de bloques del sistema de hardware a desarrollar en la Práctica # 2

4. Desarrollo de la práctica:

Para el desarrollo de la presente práctica se parte del hecho que ya se ha realizado la Práctica 1, y se emplea el mismo sistema Nios II generado en dicha práctica, para ello copiar los archivos de la práctica 1 en una nueva carpeta que será el directorio del proyecto a realizar en la práctica 2.

1. Abrir el proyecto practica_1.qpf, se procede a editar el diseño del hardware, para lo cual abrir Qsys (**Tools > Qsys**) y buscar el archivo first_system.qsys cuando se le pregunte por el archivo a abrir.

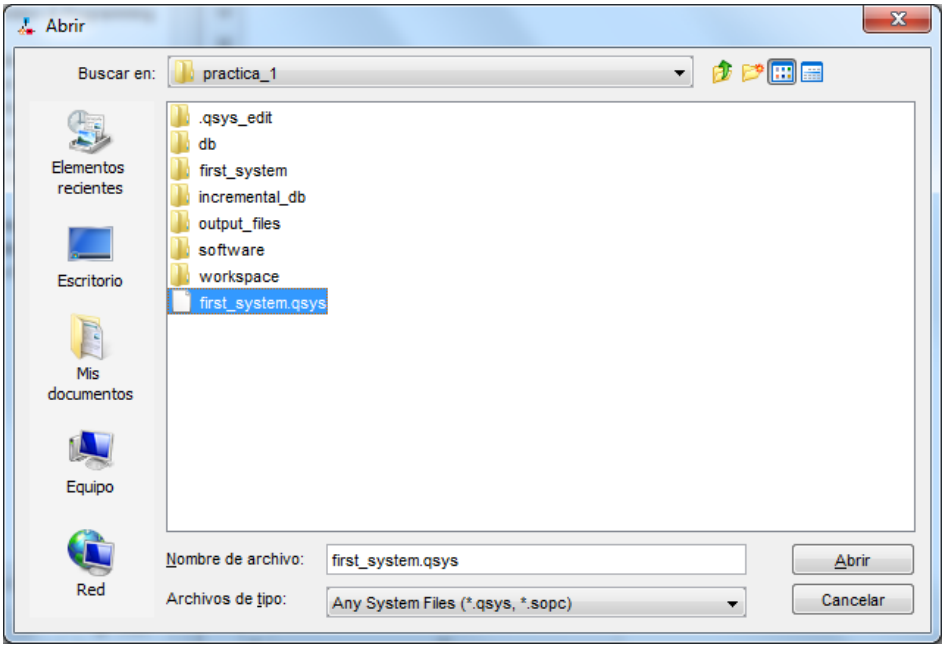


Figura 2. Selección del archivo de Qsys a editar

2. En la ventana de librería de componentes (**Component Library),** seleccionar **Memories and Memory Controllers > External Memory Interfaces > SDRAM Interfaces > SDRAM Controller** y clic en **Add.**

   En la pestaña de perfil de la memoria (Memory Profile) el parámetro *Data Width* setear a 16 bits, en la sección *Address Width* escribir 13 filas y 9 columnas, esto debido a que así está organizada la SDRAM con la que cuenta la tarjeta DE0-Nano y también ajustar los valores de los parámetros de la pestaña de tiempo como se muestra en la *Figura 4*.

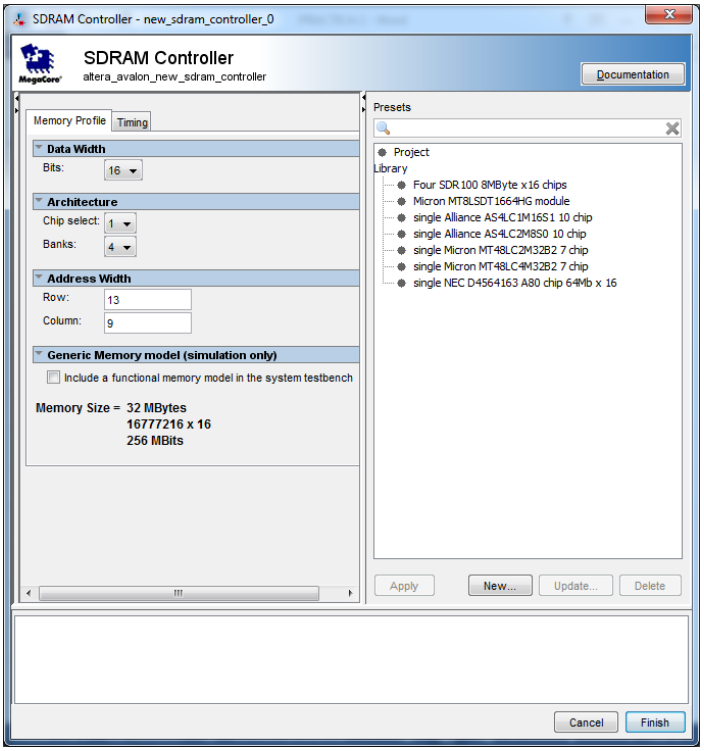


Figura 3. Ajustes del perfil de la memoria SDRAM en el controlador.

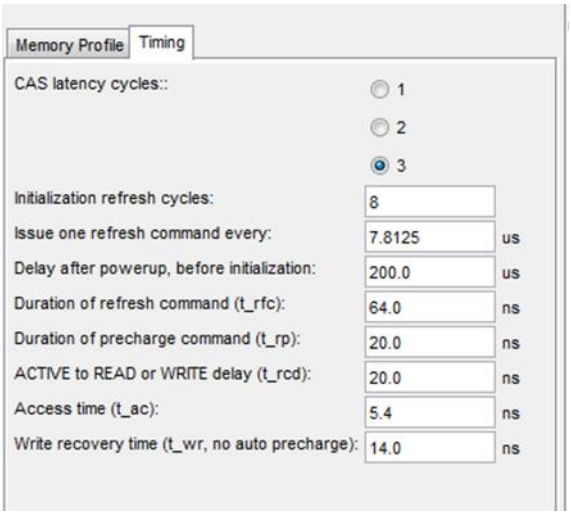


Figura 4. Ajustes de tiempo en el controlador en base a la SDRAM a utilizar.

3. Clic **Finish**, renombrar el componente como *sdram* y realizar las conexiones de la misma forma que con la on-chip memory; es decir y se exporta el SDRAM wire port con el nombre *sdram_wire*.

4. Para que el procesador Nios II pueda acceder apropiadamente al chip SDRAM, hay que evitar problemas por desplazamiento del reloj de la SDRAM que dependen de características físicas de la tarjeta DE0-Nano. Para evitar este posible problema es necesario que la señal *DRAM_CLK* adelante al reloj del sistema Nios II por 3 nanosegundos.

   Esto se puede conseguir mediante un circuito PLL (Phase-locked loop) que puede ser añadido manualmente con el MegaWizard plug-in o puede ser creado automáticamente usando el componente Clock Signals IP de los bloques de Altera University Program.

   Para añadir el componente Clock Signals IP, en la ventana de la librería de componentes seleccionar **University Program** > **Clock Signals for DE-series Board Peripherals**. Escoger DE0-Nano y dejar únicamente seleccionada la opción SDRAM. (*En caso de no contar con la librería University Program refiérase al archivo de instalación de dicha librería en la carpeta de la práctica*)

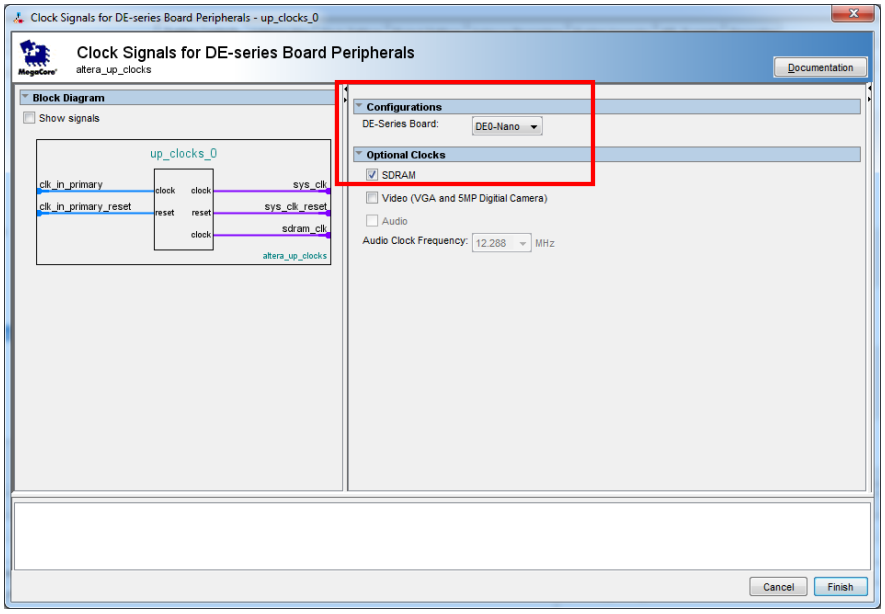


Figura 5. Adición del componente Clock Signals for DE-series Board Peripherals

Borrar todas las conexiones de la salida *clk* del componente clk_0, únicamente dejar la conexión de dicha señal con la entrada clk_in_primary del componente Clock Signal. Todos los otros componentes deben conectarse a la salida *sys_clk* en vez del reloj del sistema. Cambiar de nombre al componente Clock Signal por clocks y exportar la señal sdram_clk con el mismo nombre.

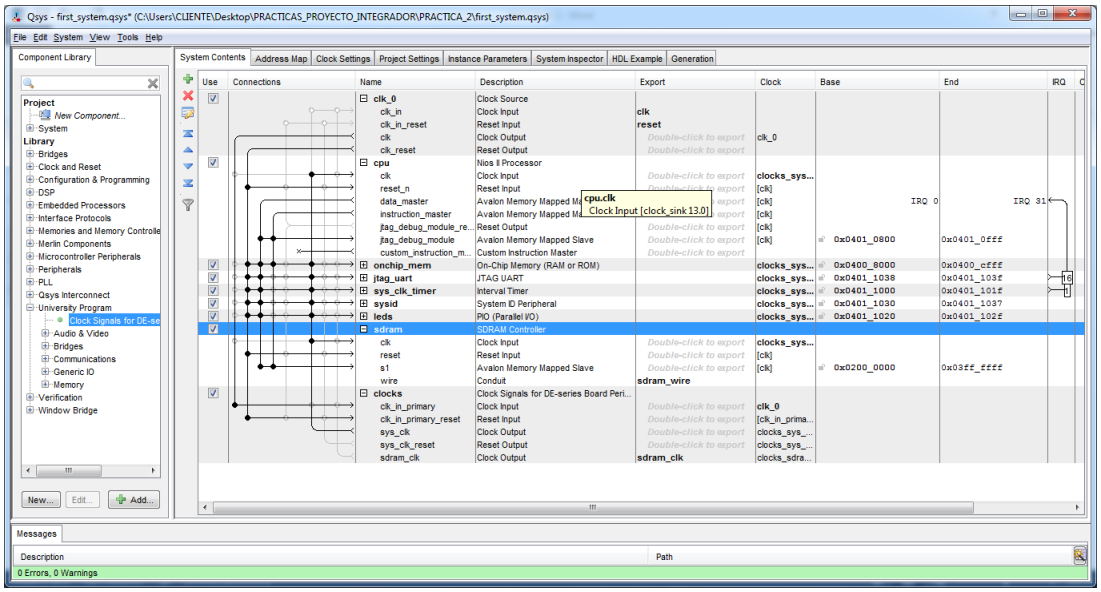


Figura 6. Conexiones de la SDRAM y del componente clocks

5. Se añaden cuatro componentes PIO (Parallel I/O), usando la ventana de librería de componentes, **Peripherals >> Microcontroller Peripherals>> Parallel I/O**. Para los datos y señales de control que se requieren para la LCD16x2, se utiliza las siguientes configuraciones:

Renombrar PIOs por:	Ancho	Tipo
lcd_data	8 bits	output
lcd_rs	1 bit	output
lcd_rw	1 bit	output
lcd_en	1 bit	output

Se realiza las conexiones del sys_clk del componente clocks y el data master del procesador a cada componente PIO. Se exportan todas estas señales desde la columna **Export** en Qsys, con el mismo nombre del componente.

6. Se repite el proceso de asignación de direcciones base, para lo cual seleccionar la pestaña **System > Assign Base Addresses** y se crea un sistema de reset general para evitar hacerlo por cada componente, **System>Create Global Reset Network.**

Figura 7. Conexiones de los componentes PIO para el control y datos de la LCD

7. Se cambia la memoria del vector de reset y el vector de excepción del procesador Nios II del sistema a la opción sdram.s1. Para ello se accede a los parámetros del componente *cpu.*

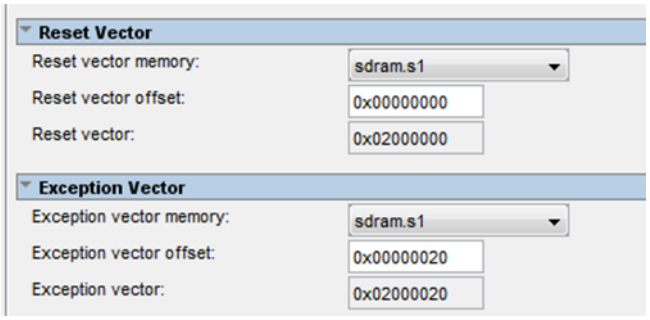


Figura 8. Asignación del vector de reset y excepción a la SDRAM

8. Guarde los cambios y genere nuevamente el sistema en Qsys, desde la pestaña **Generation**, con esto se actualiza el SOPC Information File (.sopcinfo).

9. Regresar a Quartus II y actualizar el bloque de first_system, para ello dar clic derecho sobre el bloque y seleccionar **Update Symbol or Block…** Añadir las etiquetas faltantes a los puertos correspondientes al uso del chip SDRAM, se puede realizar de manera automática al dar clic derecho y seleccionar la opción **Generate Pins for Symbol Ports** y cambiar los nombres como se muestra en la *Figura 9*.

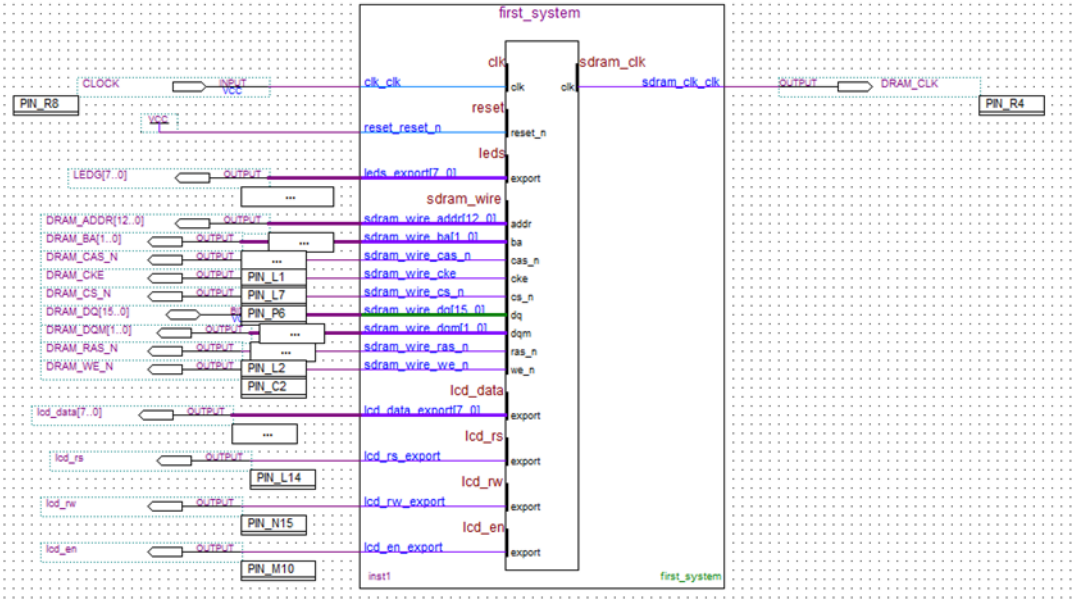


Figura 9. Instanciación del sistema de hardware mediante el diagrama de bloques

10. Para la asignación de pines nos referimos al manual del usuario de la tarjeta DE0-Nano y encontramos que se debe realizar la siguiente asignación (Standard I/O 3.3-V LVTTL):

    En el directorio del proyecto existe un archivo con la extensión .qsf, el mismo que contiene la asignación de pines, en caso de tener en digital la asignación de pines de la SDRAM puede añadirse a la parte ya existente. Una parte del archivo .qsf ya debe contener las asignaciones del desarrollo de la práctica 1

(esto se muestra *en amarillo*) y el resto del código corresponde al que debe copiarse al archivo .qsf. Para una operación apropiada debe cambiarse el estándar, en donde dice “2.5 V” cambiar por “3.3-V LVTTL”. Guarde los cambios del archivo y ciérrelo.

set_global_assignment -name STRATIX_DEVICE_IO_STANDARD "3.3-V LVTTL"

set_location_assignment PIN_R8 -to CLOCK

set_location_assignment PIN_L3 -to LEDG[7]

set_location_assignment PIN_B1 -to LEDG[6]

set_location_assignment PIN_F3 -to LEDG[5]

set_location_assignment PIN_D1 -to LEDG[4]

set_location_assignment PIN_A11 -to LEDG[3]

set_location_assignment PIN_B13 -to LEDG[2]

set_location_assignment PIN_A13 -to LEDG[1]

set_location_assignment PIN_A15 -to LEDG[0]

set_location_assignment PIN_N1 -to DRAM_ADDR[11]

set_location_assignment PIN_N2 -to DRAM_ADDR[10]

set_location_assignment PIN_P1 -to DRAM_ADDR[9]

set_location_assignment PIN_R1 -to DRAM_ADDR[8]

set_location_assignment PIN_T6 -to DRAM_ADDR[7]

set_location_assignment PIN_N8 -to DRAM_ADDR[6]

set_location_assignment PIN_T7 -to DRAM_ADDR[5]

set_location_assignment PIN_P8 -to DRAM_ADDR[4]

set_location_assignment PIN_M8 -to DRAM_ADDR[3]

set_location_assignment PIN_N6 -to DRAM_ADDR[2]

set_location_assignment PIN_N5 -to DRAM_ADDR[1]

set_location_assignment PIN_P2 -to DRAM_ADDR[0]

set_location_assignment PIN_M6 -to DRAM_BA[1]

set_location_assignment PIN_M7 -to DRAM_BA[0]

set_location_assignment PIN_L1 -to DRAM_CAS_N

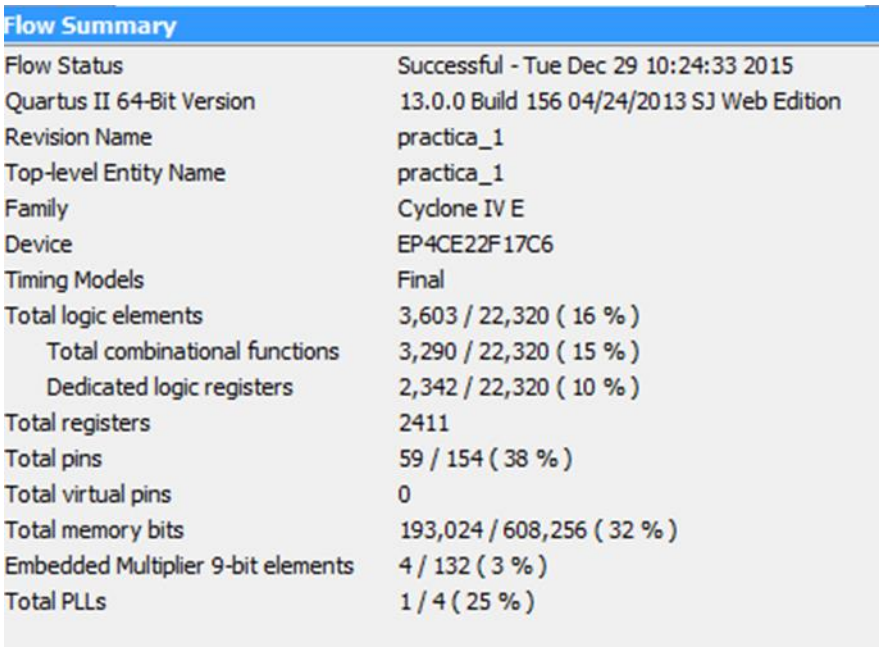
set_location_assignment PIN_L7 -to DRAM_CKE

set_location_assignment PIN_P6 -to DRAM_CS_N

```
set_location_assignment PIN_K1 -to DRAM_DQ[15]
set_location_assignment PIN_N3 -to DRAM_DQ[14]
set_location_assignment PIN_P3 -to DRAM_DQ[13]
set_location_assignment PIN_R5 -to DRAM_DQ[12]
set_location_assignment PIN_R3 -to DRAM_DQ[11]
set_location_assignment PIN_T3 -to DRAM_DQ[10]
set_location_assignment PIN_T2 -to DRAM_DQ[9]
set_location_assignment PIN_T4 -to DRAM_DQ[8]
set_location_assignment PIN_R7 -to DRAM_DQ[7]
set_location_assignment PIN_J1 -to DRAM_DQ[6]
set_location_assignment PIN_J2 -to DRAM_DQ[5]
set_location_assignment PIN_K2 -to DRAM_DQ[4]
set_location_assignment PIN_K5 -to DRAM_DQ[3]
set_location_assignment PIN_L8 -to DRAM_DQ[2]
set_location_assignment PIN_G1 -to DRAM_DQ[1]
set_location_assignment PIN_G2 -to DRAM_DQ[0]
set_location_assignment PIN_L2 -to DRAM_RAS_N
set_location_assignment PIN_C2 -to DRAM_WE_N
set_location_assignment PIN_R6 -to DRAM_DQM[0]
set_location_assignment PIN_T5 -to DRAM_DQM[1]
set_location_assignment PIN_N15 -to lcd_rw
set_location_assignment PIN_L14 -to lcd_rs
set_location_assignment PIN_M10 -to lcd_en
set_location_assignment PIN_T10 -to lcd_data[0]
set_location_assignment PIN_P11 -to lcd_data[1]
set_location_assignment PIN_N12 -to lcd_data[2]
set_location_assignment PIN_N9 -to lcd_data[3]
set_location_assignment PIN_L16 -to lcd_data[4]
set_location_assignment PIN_R16 -to lcd_data[5]
set_location_assignment PIN_P15 -to lcd_data[6]
set_location_assignment PIN_R14 -to lcd_data[7]
```

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard
CLOCK	Input	PIN_R8	3	B3_N0	PIN_R8	3.3-V LV...default)
DRAM_ADDR[12]	Output	PIN_L4	2	B2_N0	PIN_L4	3.3-V LV...default)
DRAM_ADDR[11]	Output	PIN_N1	2	B2_N0	PIN_N1	3.3-V LV...default)
DRAM_ADDR[10]	Output	PIN_N2	2	B2_N0	PIN_N2	3.3-V LV...default)
DRAM_ADDR[9]	Output	PIN_P1	2	B2_N0	PIN_P1	3.3-V LV...default)
DRAM_ADDR[8]	Output	PIN_R1	2	B2_N0	PIN_R1	3.3-V LV...default)
DRAM_ADDR[7]	Output	PIN_T6	3	B3_N0	PIN_T6	3.3-V LV...default)
DRAM_ADDR[6]	Output	PIN_N8	3	B3_N0	PIN_N8	3.3-V LV...default)
DRAM_ADDR[5]	Output	PIN_T7	3	B3_N0	PIN_T7	3.3-V LV...default)
DRAM_ADDR[4]	Output	PIN_P8	3	B3_N0	PIN_P8	3.3-V LV...default)
DRAM_ADDR[3]	Output	PIN_M8	3	B3_N0	PIN_M8	3.3-V LV...default)
DRAM_ADDR[2]	Output	PIN_N6	3	B3_N0	PIN_N6	3.3-V LV...default)
DRAM_ADDR[1]	Output	PIN_N5	3	B3_N0	PIN_N5	3.3-V LV...default)
DRAM_ADDR[0]	Output	PIN_P2	2	B2_N0	PIN_P2	3.3-V LV...default)
DRAM_BA[1]	Output	PIN_M6	3	B3_N0	PIN_M6	3.3-V LV...default)
DRAM_BA[0]	Output	PIN_M7	3	B3_N0	PIN_M7	3.3-V LV...default)
DRAM_CAS_N	Output	PIN_L1	2	B2_N0	PIN_L1	3.3-V LV...default)
DRAM_CKE	Output	PIN_L7	3	B3_N0	PIN_L7	3.3-V LV...default)
DRAM_CLK	Output	PIN_R4	3	B3_N0	PIN_R4	3.3-V LV...default)
DRAM_CS_N	Output	PIN_P6	3	B3_N0	PIN_P6	3.3-V LV...default)
DRAM_DQ[15]	Bidir	PIN_K1	2	B2_N0	PIN_K1	3.3-V LV...default)
DRAM_DQ[14]	Bidir	PIN_N3	3	B3_N0	PIN_N3	3.3-V LV...default)
DRAM_DQ[13]	Bidir	PIN_P3	3	B3_N0	PIN_P3	3.3-V LV...default)
DRAM_DQ[12]	Bidir	PIN_R5	3	B3_N0	PIN_R5	3.3-V LV...default)
DRAM_DQ[11]	Bidir	PIN_R3	3	B3_N0	PIN_R3	3.3-V LV...default)
DRAM_DQ[10]	Bidir	PIN_T3	3	B3_N0	PIN_T3	3.3-V LV...default)
DRAM_DQ[9]	Bidir	PIN_T2	3	B3_N0	PIN_T2	3.3-V LV...default)
DRAM_DQ[8]	Bidir	PIN_T4	3	B3_N0	PIN_T4	3.3-V LV...default)
DRAM_DQ[7]	Bidir	PIN_R7	3	B3_N0	PIN_R7	3.3-V LV...default)
DRAM_DQ[6]	Bidir	PIN_J1	2	B2_N0	PIN_J1	3.3-V LV...default)
DRAM_DQ[5]	Bidir	PIN_J2	2	B2_N0	PIN_J2	3.3-V LV...default)
DRAM_DQ[4]	Bidir	PIN_K2	2	B2_N0	PIN_K2	3.3-V LV...default)
DRAM_DQ[3]	Bidir	PIN_K5	2	B2_N0	PIN_K5	3.3-V LV...default)
DRAM_DQ[2]	Bidir	PIN_L8	3	B3_N0	PIN_L8	3.3-V LV...default)
DRAM_DQ[1]	Bidir	PIN_G1	1	B1_N0	PIN_G1	3.3-V LV...default)
DRAM_DQ[0]	Bidir	PIN_G2	1	B1_N0	PIN_G2	3.3-V LV...default)
DRAM_DQM[1]	Output	PIN_T5	3	B3_N0	PIN_T5	3.3-V LV...default)
DRAM_DQM[0]	Output	PIN_R6	3	B3_N0	PIN_R6	3.3-V LV...default)
DRAM_RAS_N	Output	PIN_L2	2	B2_N0	PIN_L2	3.3-V LV...default)
DRAM_WE_N	Output	PIN_C2	1	B1_N0	PIN_C2	3.3-V LV...default)
lcd_data[7]	Output	PIN_R14	4	B4_N0	PIN_R14	3.3-V LV...default)
lcd_data[6]	Output	PIN_P15	5	B5_N0	PIN_P15	3.3-V LV...default)
lcd_data[5]	Output	PIN_R16	5	B5_N0	PIN_R16	3.3-V LV...default)
lcd_data[4]	Output	PIN_L16	5	B5_N0	PIN_L16	3.3-V LV...default)
lcd_data[3]	Output	PIN_N9	4	B4_N0	PIN_N9	3.3-V LV...default)
lcd_data[2]	Output	PIN_N12	4	B4_N0	PIN_N12	3.3-V LV...default)
lcd_data[1]	Output	PIN_P11	4	B4_N0	PIN_P11	3.3-V LV...default)
lcd_data[0]	Output	PIN_T10	4	B4_N0	PIN_T10	3.3-V LV...default)
lcd_en	Output	PIN_M10	4	B4_N0	PIN_M10	3.3-V LV...default)
lcd_rs	Output	PIN_L14	5	B5_N0	PIN_L14	3.3-V LV...default)
lcd_rw	Output	PIN_N15	5	B5_N0	PIN_N15	3.3-V LV...default)
LEDG[7]	Output	PIN_L3	2	B2_N0	PIN_L3	3.3-V LV...default)
LEDG[6]	Output	PIN_B1	1	B1_N0	PIN_B1	3.3-V LV...default)
LEDG[5]	Output	PIN_F3	1	B1_N0	PIN_F3	3.3-V LV...default)
LEDG[4]	Output	PIN_D1	1	B1_N0	PIN_D1	3.3-V LV...default)
LEDG[3]	Output	PIN_A11	7	B7_N0	PIN_A11	3.3-V LV...default)
LEDG[2]	Output	PIN_B13	7	B7_N0	PIN_B13	3.3-V LV...default)
LEDG[1]	Output	PIN_A13	7	B7_N0	PIN_A13	3.3-V LV...default)
LEDG[0]	Output	PIN_A15	7	B7_N0	PIN_A15	3.3-V LV...default)

11. Compilar el proyecto para crear el archivo .sof que se descargará en la tarjeta, luego de lo cual podemos observar el reporte de compilación (*véase Figura 10*) que fue exitoso y la utilización de elementos lógicos de la FPGA. Se observa que se emplea un pll que corresponde al componente *Clock Signals for DE Series Board Peripherials.* Descargar el archivo .sof en la tarjeta DE0-Nano con la herramienta de programación (**Tools > Programmer**).



Flow Summary	
Flow Status	Successful - Tue Dec 29 10:24:33 2015
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	practica_1
Top-level Entity Name	practica_1
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	3,603 / 22,320 (16 %)
Total combinational functions	3,290 / 22,320 (15 %)
Dedicated logic registers	2,342 / 22,320 (10 %)
Total registers	2411
Total pins	59 / 154 (38 %)
Total virtual pins	0
Total memory bits	193,024 / 608,256 (32 %)
Embedded Multiplier 9-bit elements	4 / 132 (3 %)
Total PLLs	1 / 4 (25 %)

Figura 10. Reporte de compilación del sistema de hardware

12. Abrir el programa Nios II SBT desde quartus en la pestaña de herramientas (**Tools > Nios II Software Build Tools for Eclipse**) y se procede a la creación del software a correr sobre el hardware diseñado previamente, para ello crear una carpeta en el directorio del proyecto con el nombre workspace (sugerido). Crear un proyecto en blanco con la plantilla *Blank Project* (nombre sugerido: w_r_sdram), seleccionando el correspondiente archivo .sopcinfo. Al proyecto creado añadir un archivo de código fuente, para ello clic derecho sobre el proyecto w_r_sdram, seleccionar **New > File** y agregar el nombre del archivo (nombre sugerido: wr_app.c), en el mismo se escribirá el siguiente código:

```
#include <stdlib.h>
#include <stdio.h>
#include <system.h>

#define LED_ADDR 0x00009020 //espacio de memoria leds
#define SDRAM_ADDR 0x4000000

int main(void)
{
	volatile int *leds = (int*)(LED_ADDR); //asignacion del
puntero al espacio de memoria
	volatile char *mem = (char*)(SDRAM_ADDR);
	int count=255; //contador que carga el valor en el LED

	*(mem+1)=0;
	*(mem+2)=0;
	*(mem+3)=0;
	*(mem+4)=0;
	*(mem+5)=1;

	printf("Inicialmente se tiene :");
	printf("[%d],[%d],[%d],[%d],[%d]\n",

	*(mem+1),*(mem+2),*(mem+3),*(mem+4),*(mem+5));

	while (1)
	{
		usleep(100000);
			count--;
			if (count==0)
			{
				count = 255;
			}
			printf("posicion : %d  Valor  --> %d ==>
",mem+255-count,count);
			if (count>249)
			*(mem+255-count)=count;
			*(leds) = count;
		printf("[%d],[%d],[%d],[%d],[%d]\n",

	*(mem+1),*(mem+2),*(mem+3),*(mem+4),*(mem+5));
	}
	return 0;
}
```

En el código mostrado debe tenerse en cuenta que correspondan las direcciones de los punteros tanto a los leds como a la SDRAM, esto se puede ver en Qsys en la columna de direcciones base. Además también se encuentra en la carpeta del proyecto w_r_sdram_bsp en el archivo system.h.

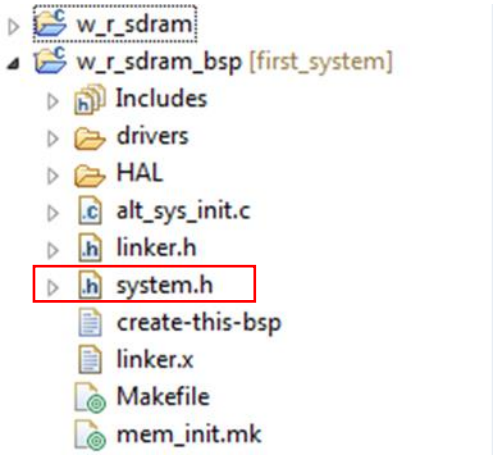


Figura 11. Ubicación del archivo system.h que contiene información del sistema

13. Dar doble clic al archivo system.h, en el mismo se tienen las características del hardware, y se busca la parte del código que contiene la información de los leds y de la SDRAM, donde se visualizan sus direcciones base. En caso de que sea diferente la dirección del archivo de código fuente cambiar por la dirección del archivo system.h. Guardar los cambios del archivo wr_app.c

```
/*
 * sdram configuration
 *
 */

#define ALT_MODULE_CLASS_sdram altera_avalon_new_sdram_controller
#define SDRAM_BASE 0x4000000
#define SDRAM_CAS_LATENCY 3
```

Figura 12. Dirección base del componente llamado sdram

14. Construir el proyecto, para ello clic derecho sobre el proyecto *w_r_sdram* y seleccionar **Build Project**, una vez finalizado este proceso, nuevamente clic derecho sobre el proyecto de aplicación y seleccionar **Run As > Nios II Hardware**, una vez que corre el programa se muestra en consola y en los leds el decremento del contador binario de 8 bits y se escribe y lee en la SDRAM los 5 primeros números del contador después del primer decremento (254, 253, 252, 251, 250). Para detalles sobre el chip SDRAM revisar el anexo de la presente práctica.

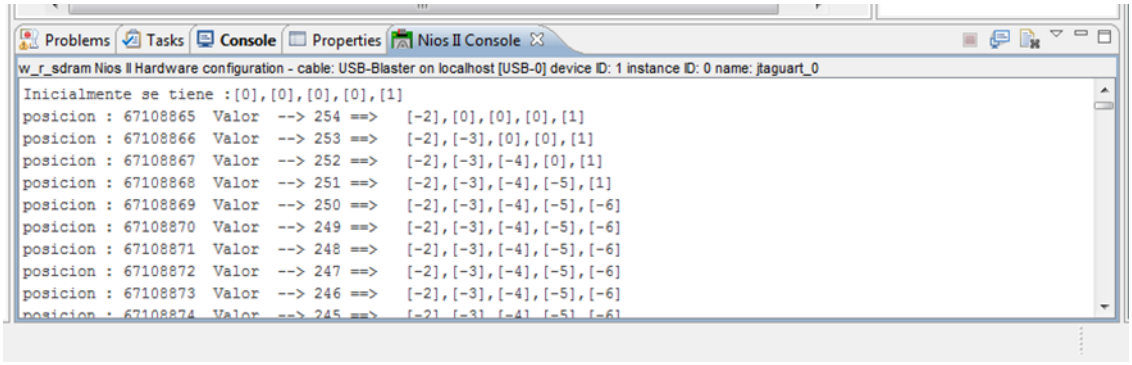


Figura 13. Consola de Nios II SBT, mostrando la lectura y escritura en la SDRAM

15. Se crea otro proyecto, para controlar la LCD 16x2, a partir de la plantilla **Blank Project** (nombre sugerido: sdram_lcd), seleccionando el correspondiente archivo .sopcinfo (el mismo de la prueba anterior). Al proyecto creado añadir un archivo main.c, en el mismo se escribirá el siguiente código, luego construir y compilar el proyecto.

```
#include <stdio.h>
#include <string.h>
#include <system.h>
#include "alt_types.h"

void Lcd_Init();
void Lcd_Clear();
void Pulso_EN();
void ProcesarComando(char comando);
void ProcesarDato(char letra);
void Lcd_Write(char* cadena);
void Lcd_Erase();
void Lcd_Linea1();
void Lcd_Linea2();
/* void Lcd_Linea3();
void Lcd_Linea4();*/
volatile int *LCD_Dato = (volatile int *)LCD_DATA_BASE;
volatile int *LCD_En = (volatile int *)LCD_EN_BASE;
volatile int *LCD_Rs = (volatile int *)LCD_RS_BASE;
volatile int *LCD_Wr = (volatile int *)LCD_RW_BASE;
```

```
int main(){
	printf("Probando LCD\n");
	*LCD_Wr = 0;
	Lcd_Init();
	Lcd_Clear();
	Lcd_Linea1();
	ProcesarDato(' ');
	ProcesarDato(' ');
	ProcesarDato(' ');
	ProcesarDato(' ');
	ProcesarDato(' ');
	ProcesarDato(' ');
	ProcesarDato('H');
	ProcesarDato('O');
	ProcesarDato('L');
	ProcesarDato('A');
	Lcd_Linea2();
	Lcd_Write("   SDRAM_LCD");
	/*Lcd_Linea3();
	ProcesarDato('H');
	ProcesarDato('O');
	ProcesarDato('L');
	ProcesarDato('A');
	Lcd_Linea4();
	Lcd_Write("Hola");*/
	while(1);

	return 0;
}
void Lcd_Init(){
	ProcesarComando(0x3C); // Datos de 8 bits, Modo 2 lineas,
caracter 5x10
	ProcesarComando(0x0C); // Enciende la LCD
	ProcesarComando(0x06); // Desplaza cursor hacia la derecha
```

```
}
void Lcd_Clear(){
	ProcesarComando(0x01); // Borra la LCD
	usleep(1640);
}
void Pulso_EN(){ // Se genera el pulso de EN para transferir los datos a la LCD
	*LCD_En=1;
	usleep(1);
	*LCD_En=0;
}
void ProcesarComando(char comando){
	*LCD_Dato=0x00;
	*LCD_Rs=0;
	*LCD_Dato=comando;
	Pulso_EN();
	usleep(50);
}
void ProcesarDato(char letra){
	*LCD_Dato=0x00;
	*LCD_Rs=1;
	*LCD_Dato=letra;
	Pulso_EN();
	usleep(50);
}
void Lcd_Write(char* cadena){
	int i=0;
	for(i=0; i<strlen(cadena); i++){
		ProcesarDato(cadena[i]);
		usleep(12500);
	}
}
void Lcd_Linea1(){
	ProcesarComando(0x80);
}
```

```
void Lcd_Linea2(){
	ProcesarComando(0xC0);
}
/*void Lcd_Linea3(){
	ProcesarComando(0x94);
}
void Lcd_Linea4(){
	ProcesarComando(0xD4);
}*/
```

16. En la consola de Nios II SBT se muestra la frase “Probando LCD”, mientras en la pantalla LCD el mensaje HOLA en la primera línea y SDRAM_LCD en la segunda línea. Para un mayor entendimiento del código empleado, revisar el anexo de la práctica que contiene la información sobre el control de la LCD 16x2. Como se observa en el código ciertas líneas se encuentran comentadas y son para el uso de una LCD 20x4.

Figura 14. Consola de Nios II SBT, mostrando un mensaje durante la ejecución del código para la escritura en la LCD.

Figura 15. Escritura en la LCD 16x2

PRÁCTICA # 3

TEMA: *Adquisición de datos de los canales del chip convertidor ADC integrado en la tarjeta DE0-Nano.*

1. Objetivos:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Entender y utilizar el chip ADC128S022 integrado en la tarjeta de desarrollo DE0-Nano.
- Emplear el controlador del ADC de la tarjeta DE0-Nano por medio de la herramienta de desarrollo de hardware, *Qsys.*
- Ejecutar un software sobre el hardware desarrollado utilizando la herramienta *'Nios II Software Build for Eclipse',* para verificar la correcta operación del chip ADC.

2. Requisitos mínimos:

- Tener instalado Quartus II y Nios II EDS (versión 13.0).
- Conocimientos básicos de Quartus II y programación en VHDL y C.
- Haber culminado satisfactoriamente la Práctica # 2.
- Lectura y comprensión del archivo: '*FUNDAMENTACIÓN_TEÓRICA_PRÁCTICA_ADC'*

3. Breve explicación de la práctica:

Se desarrolla un sistema de hardware que utiliza el procesador embebido de Altera, Nios II estándar en conjunto con otros módulos *(véase Figura 1)*, conectándose por medio de una interconexión llamada *Avalon switch fabric,* para formar un sistema que

se implementa en el chip de la FPGA Cyclone IV EP4CE22F17C6 utilizando el programa de *Quartus II*.

Entre los componentes utilizados se encuentra el ***DE0-Nano ADC Controller***, el mismo que establece una interfaz entre el procesador y el chip ADC128S002 de la tarjeta DE0-Nano. Todas las partes del sistema Nios II son definidas usando lenguaje de descripción de hardware que se genera de forma automática utilizando la herramienta de *Qsys* para implementar el sistema deseado mediante la selección de los componentes requeridos, especificación de sus parámetros, interconexión de los mismos mediante una matriz de conexiones que ofrece la herramienta y asignación de direcciones de memoria de cada parte del sistema Nios II.

Sobre el diseño de hardware desarrollado se corre un programa en lenguaje C creado en '*Nios II Software Build Tools for Eclipse'*. Se evidencia el correcto desarrollo de la práctica mostrando en la consola el voltaje del canal seleccionado con los interruptores de la tarjeta y con la visualización en los 8 LEDs incorporados en la tarjeta DE0-Nano de los 8 bits más significativos de la conversión del ADC.

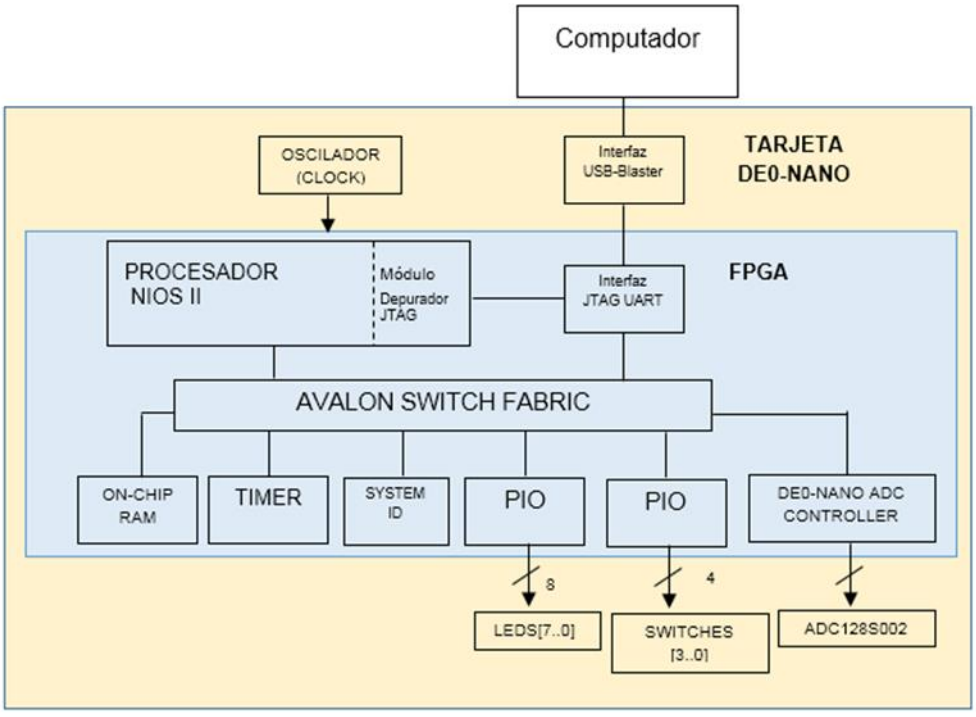


Figura 1. Diagrama de bloques del sistema de hardware a desarrollar en la Práctica # 3

4. Desarrollo de la práctica:

1. Abrir el programa Quartus II y crear un proyecto nuevo (*Nombre del proyecto sugerido: practica_adc)* con la familia *Cyclone IV E, dispositivo EP4CE22F17C6* y guardarlo en un nuevo directorio (practica_3) donde se almacenarán todos los archivos que se generen en el desarrollo de la práctica. ***Nota****. En la ventana EDA tool settings, seleccione <None> en todas las opciones de la columna Tool Name.*

2. Una vez creado el proyecto en Quartus, seleccionar **Tools > Qsys**.

3. Añadir los componentes básicos del sistema desde la librería, para ello seleccionar y agregar los que se describen en la *Tabla 1* con sus respectivas configuraciones:

Componente	**Renombrar componentes**	**Configuración**
Nios II Processor	cpu	Nios II/s
On-Chip Memory (RAM or ROM)	onchip_mem	Total memory size = 30720 bytes
System ID Peripherial	sysid	default
Interval Timer	timer	default
PIO (Parallel I/O)	pio_leds	Width = 8 bits Output
JTAG UART	jtag_uart	default
PIO (Parallel I/O)	pio_switches	Width = 4 bits Input

Tabla 1. Componentes básicos del sistema de hardware de la práctica # 3

En caso de requerir procedimientos más detallados para agregar los componentes antes mencionados refiérase a la práctica 1.

4. Añadir el controlador ADC, para ello en la librería de componentes seguir la ruta **University Program** > **Generic IO > DE0-Nano ADC Controller** y dar doble clic (o en su defecto **Add**).

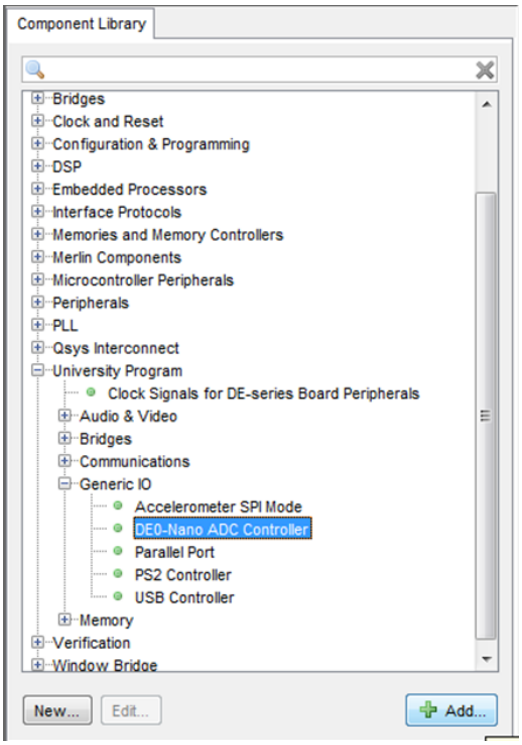


Figura 2. Adición del componente controlador del ADC de la DE0-Nano

Dejar las características por defecto del controlador que corresponden a la máxima frecuencia que permite el ADC128S002, 3.2 MHz y a 8 canales que tiene disponible.

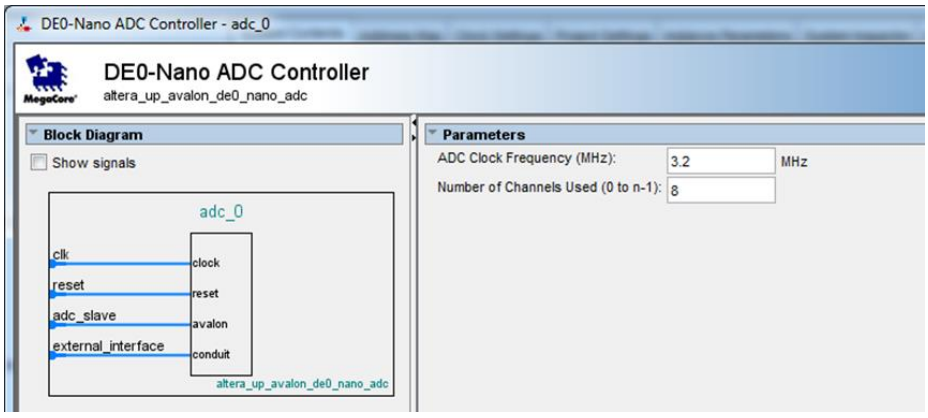


Figura 3. Configuración de los parámetros del controlador ADC

5. Recordar que los errores que se visualizan se deben a la falta de conexiones, de direcciones base a los componentes y asignaciones de vectores de memoria de reset y de excepción en el procesador. Las conexiones a realizar se muestran en la *Figura 4*.

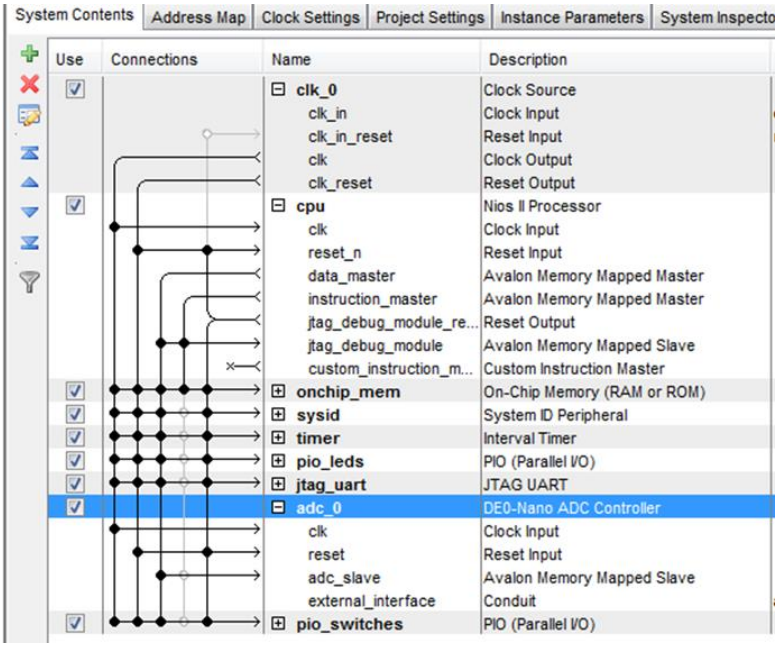


Figura 4. Conexiones entre los componentes del sistema de hardware

6. Se exportan las señales de los componentes a las que posteriormente se le realizarán asignación de pines de la FPGA para ello en la columna **Export** se da doble clic sobre las señales a exportar y se cambia los nombres como se muestra en la *Figura 5*.

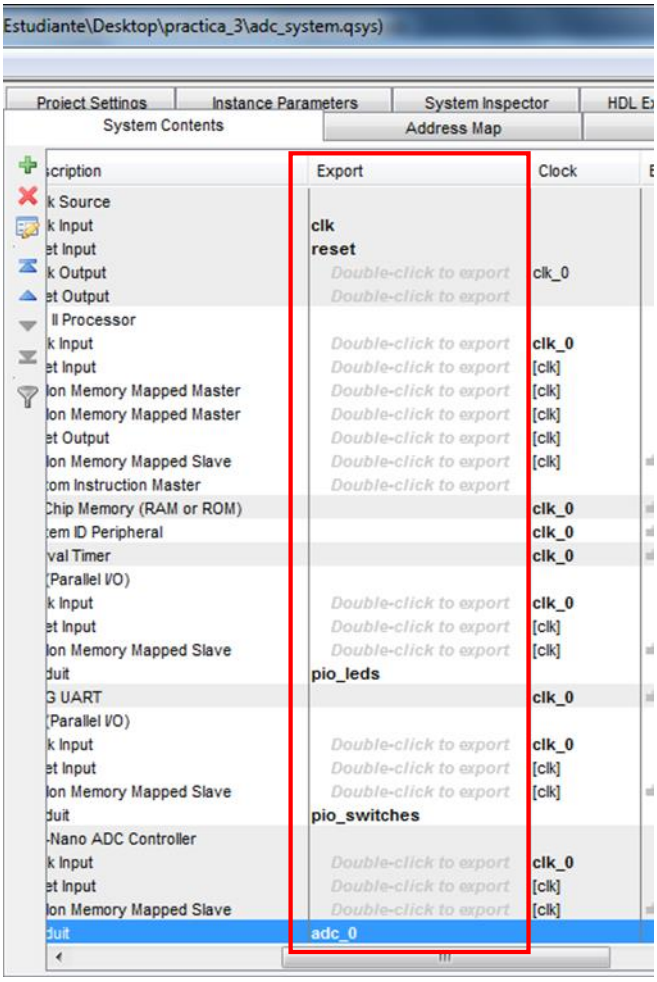


Figura 5. Señales exportadas del sistema

7. Se debe añadir las solicitudes de interrupción o IRQs (Interrupt Request). En la columna IRQ, conectar el Procesador Nios II al JTAG UART y al Interval Timer. Clic en el valor IRQ del jtag_uart y escribir 16, de manera similar escribir 1 para el valor IRQ del componente sys_clk_timer.

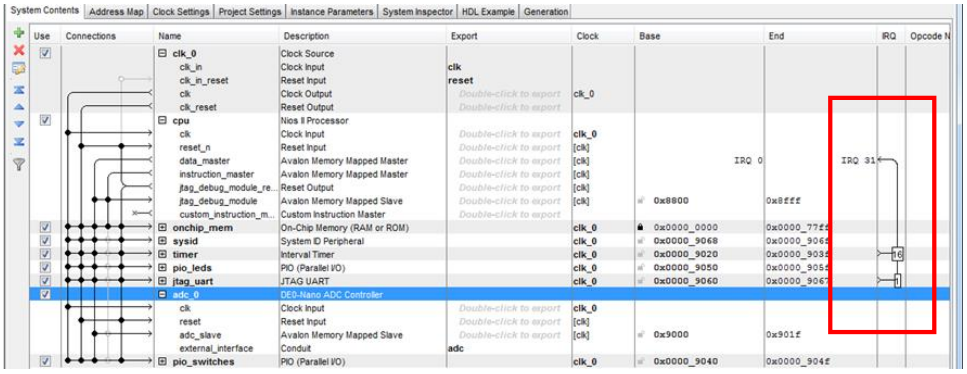


Figura 6. Asignación de la prioridad de interrupciones

8. Añadir las direcciones base, como sugerencia establecer manualmente la dirección base de la on-chip memory a 0x00000000 y fijarlo a ese valor seleccionando el candado ubicado a lado izquierdo de la dirección, para todos los demás componentes generar las direcciones de forma automática mediante la opción *Assign Base Addresses* en la pestaña System.

9. Guardar y llenar el nombre de archivo (sugerido adc_system). Clic en la pestaña **Generation** y seleccionar None tanto en la sección **Simulation** como en **Testbench System.** Clic **Generate,** al finalizar clic en **Close,** cerrar Qsys y regresar a Quartus II.
10. Se procede a añadir el archivo de extensión .qip generado en Qsys, que contiene la información del sistema Nios II creado. En la ventana de **Project Navigator**, dar doble clic sobre la carpeta **Files**, y buscar el archivo que se encuentra en la ruta: *<nombre del directorio>/adc_system/synthesis/adc_system.qip* (*Verificar que se esté mostrando todos los archivos o los de extensión .qip al buscar*)

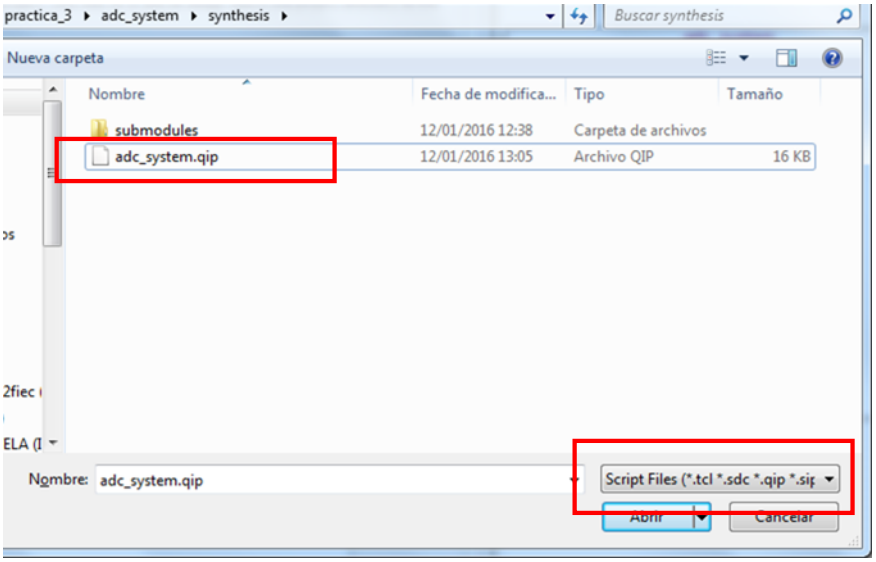


Figura 7. Adición del archivo .qip al proyecto en quartus

11. Se procede a instanciar el módulo del sistema creado en Qsys, para ello crear un nuevo *Block Diagram / Schematic File*, y añadir el bloque adc_system. Agregar los terminales de entrada y salida, con sus respectivas etiquetas. Guardar el archivo (practica_adc).

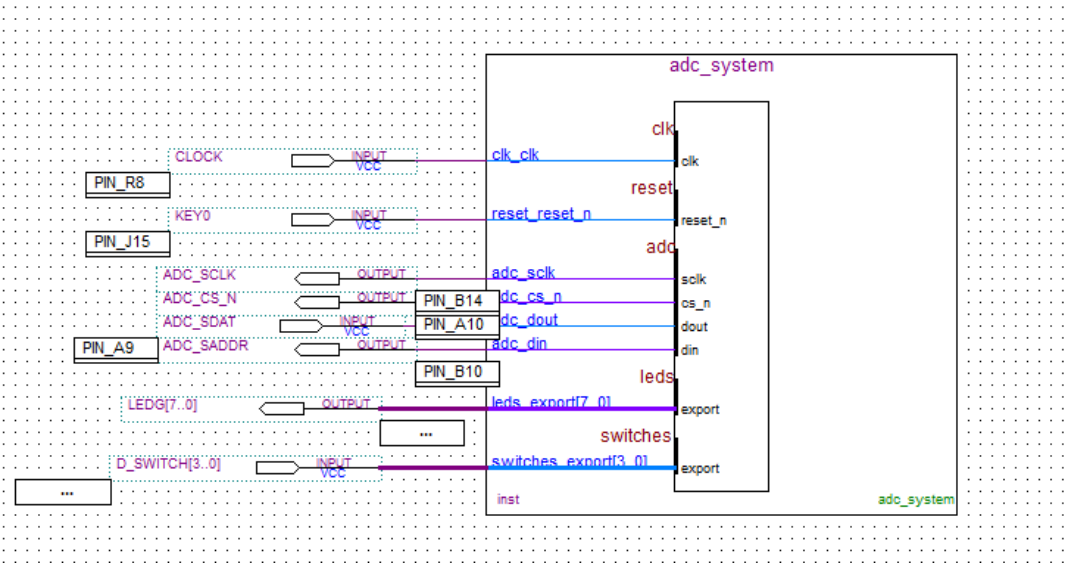


Figura 8. Adición de los pines al nuevo bloque adc_system

12. En el menú **Processing** seleccionar **Start** > **Start Analysis & Elaboration,** con lo cual al revisar el **Pin Planner** observamos listadas las entradas y salidas creadas.

13. Para la asignación de pines nos referimos al manual del usuario de la tarjeta DE0-Nano y encontramos que se debe realizar la siguiente asignación (Standard I/O 3.3-V LVTTL), para emplear los switches, leds, clock y adc de la tarjeta.

    En el directorio del proyecto existe un archivo con la extensión .qsf, el mismo que contiene la asignación de pines, en caso de tener en digital la asignación de pines puede añadirse a la parte ya existente. Se puede copiar el siguiente código en el archivo .qsf ubicado en el directorio similar a la práctica 2. Para una operación apropiada debe cambiarse el estándar, en donde dice “2.5 V” cambiar por “3.3-V LVTTL”. Guarde los cambios del archivo y ciérrelo.

```
set_global_assignment -name STRATIX_DEVICE_IO_STANDARD "3.3-V LVTTL"
set_location_assignment PIN_R8 -to CLOCK
set_location_assignment PIN_L3 -to LEDG[7]
set_location_assignment PIN_B1 -to LEDG[6]
set_location_assignment PIN_F3 -to LEDG[5]
set_location_assignment PIN_D1 -to LEDG[4]
set_location_assignment PIN_A11 -to LEDG[3]
set_location_assignment PIN_B13 -to LEDG[2]
set_location_assignment PIN_A13 -to LEDG[1]
set_location_assignment PIN_A15 -to LEDG[0]
set_location_assignment PIN_J15 -to KEY0
set_location_assignment PIN_A10 -to ADC_CS_N
set_location_assignment PIN_B10 -to ADC_SADDR
set_location_assignment PIN_B14 -to ADC_SCLK
set_location_assignment PIN_A9 -to ADC_SDAT
set_location_assignment PIN_M15 -to D_SWITCH[3]
set_location_assignment PIN_B9 -to D_SWITCH[2]
set_location_assignment PIN_T8 -to D_SWITCH[1]
set_location_assignment PIN_M1 -to D_SWITCH[0]
```

Named: * Edit:

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved
ADC_CS_N	Output	PIN_A10	7	B7_N0	PIN_A10	3.3-V LV...default)	
ADC_SADDR	Output	PIN_B10	7	B7_N0	PIN_B10	3.3-V LV...default)	
ADC_SCLK	Output	PIN_B14	7	B7_N0	PIN_B14	3.3-V LV...default)	
ADC_SDAT	Input	PIN_A9	7	B7_N0	PIN_A9	3.3-V LV...default)	
CLOCK	Input	PIN_R8	3	B3_N0	PIN_R8	3.3-V LV...default)	
D_SWITCH[3]	Input	PIN_M15	5	B5_N0	PIN_M15	3.3-V LV...default)	
D_SWITCH[2]	Input	PIN_B9	7	B7_N0	PIN_B9	3.3-V LV...default)	
D_SWITCH[1]	Input	PIN_T8	3	B3_N0	PIN_T8	3.3-V LV...default)	
D_SWITCH[0]	Input	PIN_M1	2	B2_N0	PIN_M1	3.3-V LV...default)	
KEY0	Input	PIN_J15	5	B5_N0	PIN_J15	3.3-V LV...default)	
LEDG[7]	Output	PIN_L3	2	B2_N0	PIN_L3	3.3-V LV...default)	
LEDG[6]	Output	PIN_B1	1	B1_N0	PIN_B1	3.3-V LV...default)	
LEDG[5]	Output	PIN_F3	1	B1_N0	PIN_F3	3.3-V LV...default)	
LEDG[4]	Output	PIN_D1	1	B1_N0	PIN_D1	3.3-V LV...default)	
LEDG[3]	Output	PIN_A11	7	B7_N0	PIN_A11	3.3-V LV...default)	
LEDG[2]	Output	PIN_B13	7	B7_N0	PIN_B13	3.3-V LV...default)	
LEDG[1]	Output	PIN_A13	7	B7_N0	PIN_A13	3.3-V LV...default)	
LEDG[0]	Output	PIN_A15	7	B7_N0	PIN_A15	3.3-V LV...default)	

Figura 9. Pines asignados a cada señal exportada

14. Compilar el proyecto para crear el archivo .sof que se descargará en la tarjeta, luego de lo cual podemos observar el reporte de compilación que fue exitoso y la utilización de elementos lógicos de la FPGA.

Flow Summary	
Flow Status	Successful - Tue Jan 12 13:29:06 2016
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Full Version
Revision Name	practica_adc
Top-level Entity Name	practica_adc
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	2,782 / 22,320 (12 %)
Total combinational functions	2,447 / 22,320 (11 %)
Dedicated logic registers	1,880 / 22,320 (8 %)
Total registers	1880
Total pins	18 / 154 (12 %)
Total virtual pins	0
Total memory bits	291,456 / 608,256 (48 %)
Embedded Multiplier 9-bit elements	4 / 132 (3 %)
Total PLLs	0 / 4 (0 %)

Figura 10. Reporte de compilación del sistema de hardware

15. Descargar el archivo .sof en la tarjeta DE0-Nano con la herramienta de programación (**Tools > Programmer**).

16. Abrir el programa Nios II SBT desde quartus en la pestaña de herramientas (**Tools > Nios II Software Build Tools for Eclipse**) y se procede a la creación

del software a correr sobre el hardware diseñado previamente, para ello crear una carpeta en el directorio del proyecto con el nombre workspace (sugerido). Crear un proyecto en blanco (nombre sugerido: practica_adc), seleccionando el correspondiente archivo .sopcinfo. Al proyecto creado añadir un archivo main.c, en el mismo se escribirá el siguiente código:

```
#include <stdlib.h>
#include <stdio.h>
#include <system.h>

int main (void) {

      /* Declare volatile pointers to I/O registers (volatile means
that IO load
       * and store instructions will be used to access these pointer
locations,
       * instead of regular memory loads and stores)
      */
      volatile int * ADC = (int *) 0x00009000;
      volatile int * Dip_Switch = (int *) 0x00009040;
      volatile int * Green_LEDs = (int *) 0x00009050;

      unsigned int data;
      unsigned int data1;
      unsigned int channel;

      while (1){
            *(ADC) = 0;                //Get the ADC to start
reading the analog data
            channel = *(Dip_Switch); //Read the Switches to select
a channel
            channel = channel % 8;
            data = *(ADC+channel);     //Get the value of the
selected channel
            data1=(data*3.3)/4095;    //Convert measure (3.3 V
supply power, 4095=2^12 max value)
            printf("El valor de voltaje del canal %d es -->  %d
Voltios \n",channel,data1);
            usleep(100000);
            data = data/16;            //Ignore the lowest 4 bits
            *(Green_LEDs) = data;      //Display the value on the
LEDs
      }
      return 0;
}
```

En el código mostrado debe tenerse en cuenta que correspondan las direcciones de los punteros tanto a los leds, switches y el ADC según el hardware creado, esto se puede ver en Qsys en la columna de direcciones

base. Además también se encuentra en la carpeta del proyecto practica_adc_bsp en el archivo system.h.

17. Clic derecho sobre el proyecto *practica_adc_bsp* y seleccionar **Properties.** En la opción *Nios II BSP Properties* se habilitan únicamente las opciones *Reduced device drivers* y *Small C library,* las demás no se seleccionan (*véase Figura 11*).

    Esto se realiza para minimizar el consumo de memoria del software debido a que nuestro sistema tiene sólo 30 KB de memoria y parte de la misma como se observa en el reporte de compilación se usó para guardar el sistema de hardware.

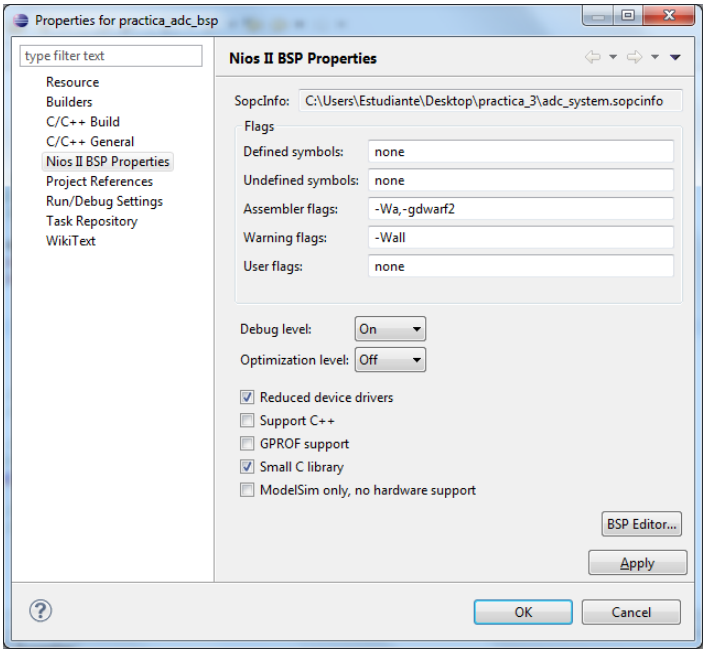


Figura 11. Ajuste de las propiedades del proyecto BSP

18. Clic derecho sobre el proyecto *practica_adc* y seleccionar **Build Project**, al finalizar se muestra un mensaje de finalizado. Nuevamente, clic derecho sobre el proyecto *contador* y seleccionar **Run As** > **Nios II Hardware.**

    Una vez que el programa se ha ejecutado se muestra en consola el voltaje del canal seleccionado y en los 8 leds de la tarjeta los 8 bits más significativos de la conversión.

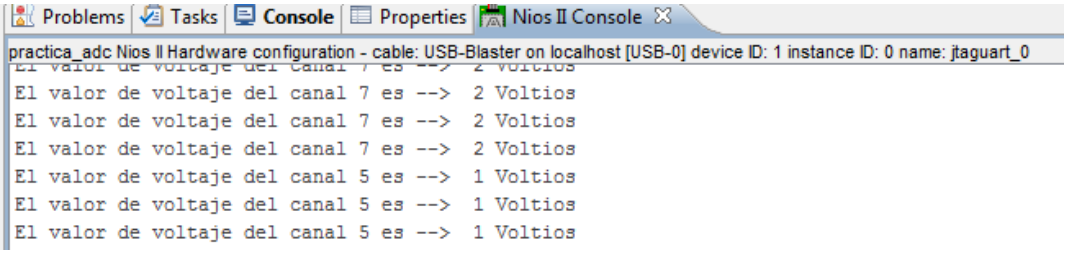


Figura 12. Consola de Nios II SBT mostrando el voltaje medido en el canal seleccionado

Para un mayor entendimiento del código empleado, revisar el anexo de la práctica que contiene la información sobre los registros del controlador del chip ADC integrado en la DE0-Nano.

PRÁCTICA # 4

TEMA: *Lectura de los datos medidos por el chip ADXL345, acelerómetro de 3 ejes con hasta 13 bits de resolución, integrado en la tarjeta DE0-Nano.*

1. Objetivos:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Entender y explicar el uso de acelerómetros para el sensado del grado de inclinación.
- Emplear los componentes necesarios de Qsys para controlar el acelerómetro ADXL345 de la tarjeta DE0-Nano.
- Ejecutar una aplicación de software sobre el hardware desarrollado utilizando *'Nios II Software Build for Eclipse',* para acceder a los registros que contienen los valores medidos por el acelerómetro ADXL345.

2. Requisitos mínimos:

- Tener instalado Quartus II y Nios II EDS (versión 13.0).
- Conocimientos básicos de Quartus II y programación en VHDL y C.
- Haber culminado satisfactoriamente la Práctica # 2.
- Lectura y comprensión del archivo: '*FUNDAMENTACIÓN_TEÓRICA_PRÁCTICA_ACELERÓMETRO*'

3. Breve explicación de la práctica:

Se desarrolla un sistema de hardware que incluye un procesador embebido de Altera Nios II en conjunto con otros módulos, entre los cuales se encuentra el componente de *propiedad intelectual IP: Acelerometer SPI Mode (véase figura 1)*, para el control del chip ADXL345, un acelerómetro de tres ejes contenido en la tarjeta de desarrollo DE0-Nano. Todos los componentes se conectan por medio de una interconexión llamada *Avalon switch fabric.* De esta manera se forma un sistema que luego se implementa en el chip FPGA utilizando el programa de *Quartus II*.

Todas las partes del sistema basado en el procesador Nios II, son definidas usando lenguaje de descripción de hardware utilizando la herramienta *Qsys* para implementar el sistema deseado simplemente seleccionando los componentes requeridos y especificando los parámetros necesarios. Como memoria del programa y de datos se utiliza el chip SDRAM embebido en la tarjeta DE0-Nano para evitar desbordes de memoria que pueden darse en caso de utilizarse la On-Chip Memory.

Sobre el diseño de hardware desarrollado se corre un programa creado *en Nios II Software Build Tools for Eclipse,* que permite mostrar en consola los valores medidos por el acelerómetro en el eje X y los ocho bits menos significativos de la lectura se muestran en los leds de la tarjeta. Se explica además como se da la lectura de los otros ejes (Y, Z).

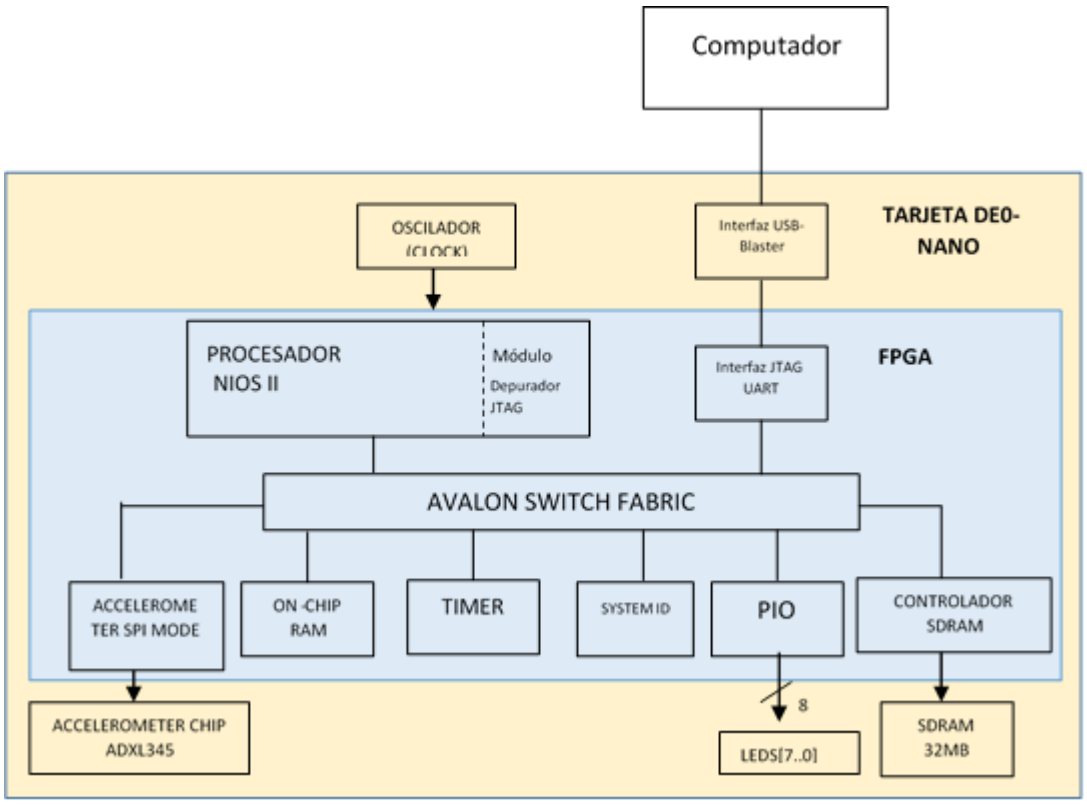


Figura 1. Diagrama de bloques del sistema de hardware a desarrollar en la Práctica # 4

4. Desarrollo de la práctica:

1. Abrir el programa Quartus II y crear un proyecto nuevo (*Nombre del proyecto sugerido: practica_accelerometer)* con la familia *Cyclone IV E, dispositivo EP4CE22F17C6* y guardarlo en un nuevo directorio (practica_4) donde se almacenarán todos los archivos que se generen en el desarrollo de la práctica. ***Nota****. En la ventana EDA tool settings, seleccione <None> en todas las opciones de la columna Tool Name.*

2. Una vez creado el proyecto en Quartus, seleccionar **Tools > Qsys**.

3. Añadir los componentes básicos del sistema desde la librería, para ello seleccionar y agregar los que se describen en la *Tabla 1* con sus respectivas configuraciones:

Componente	**Renombrar componentes**	**Configuración**
Nios II Processor	cpu	Nios II/s
On-Chip Memory (RAM or ROM)	onchip_mem	default
System ID Peripherial	sysid	default
JTAG UART	jtag_uart	default
Interval Timer	timer	default
PIO (Parallel I/O)	switches	Width = 4 bits Input
PIO (Parallel I/O)	LEDs	Width = 8 bits Ouput
SDRAM Controller	sdram	Data Width = 16 bits Chip Select=1, Banks=4 Row= 13, Columns=9
Clock Signals for DE-series Board Peripherals	clocks	DE-Series Board = DE0-Nano Optional Clocks → Sólo señalar SDRAM

Tabla 1. Componentes básicos del sistema de hardware de la práctica # 4

En caso de requerir procedimientos más detallados para agregar los componentes antes mencionados refiérase a la práctica 1 que contiene parte introductoria al uso de Qsys para el desarrollo de sistemas embebidos en el procesador Nios II y a la práctica 2 en el uso de la memoria SDRAM.

4. Añadir el controlador del acelerómetro, para ello en la librería de componentes seguir la ruta **University Program** > **Generic IO > Accelerometer SPI Mode** y dar doble clic (o en su defecto **Add**).

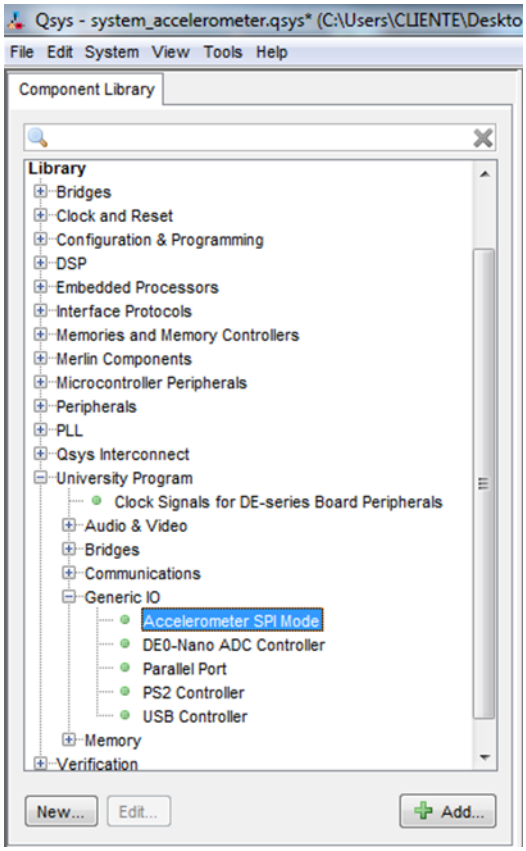


Figura 2. Adición del componente controlador del acelerómetro de la DE0-Nano

Al agregar este componente observamos que no hay parámetros a modificar, por lo que directamente damos clic en **Finish**.

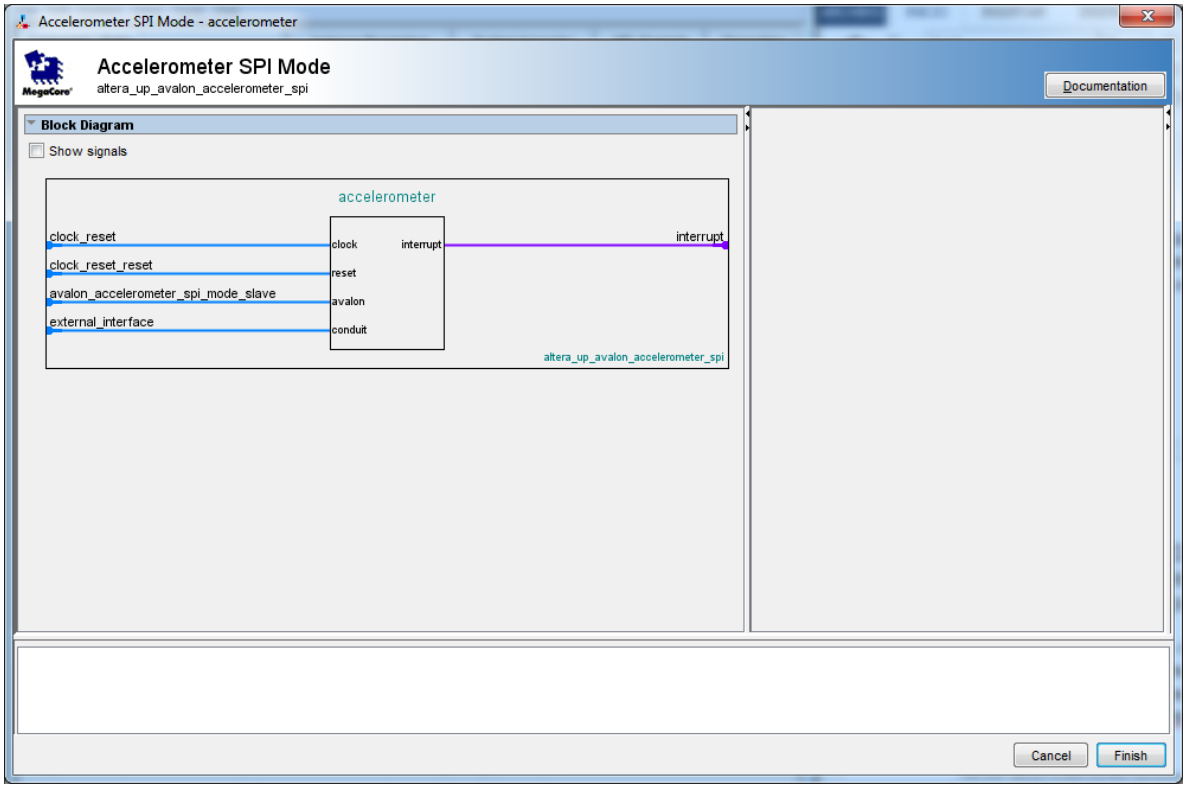


Figura 3. Ventana de ajustes del componente de control del acelerómetro

5. Recordar que los errores que se visualizan se deben a la falta de conexiones, de direcciones base a los componentes y asignaciones de vectores de memoria de reset y de excepción en el procesador (*Asignar estos vectores a la SDRAM, recordar que se asigna en los parámetros del procesador*). Las conexiones a realizar se muestran en la *Figura 4*.

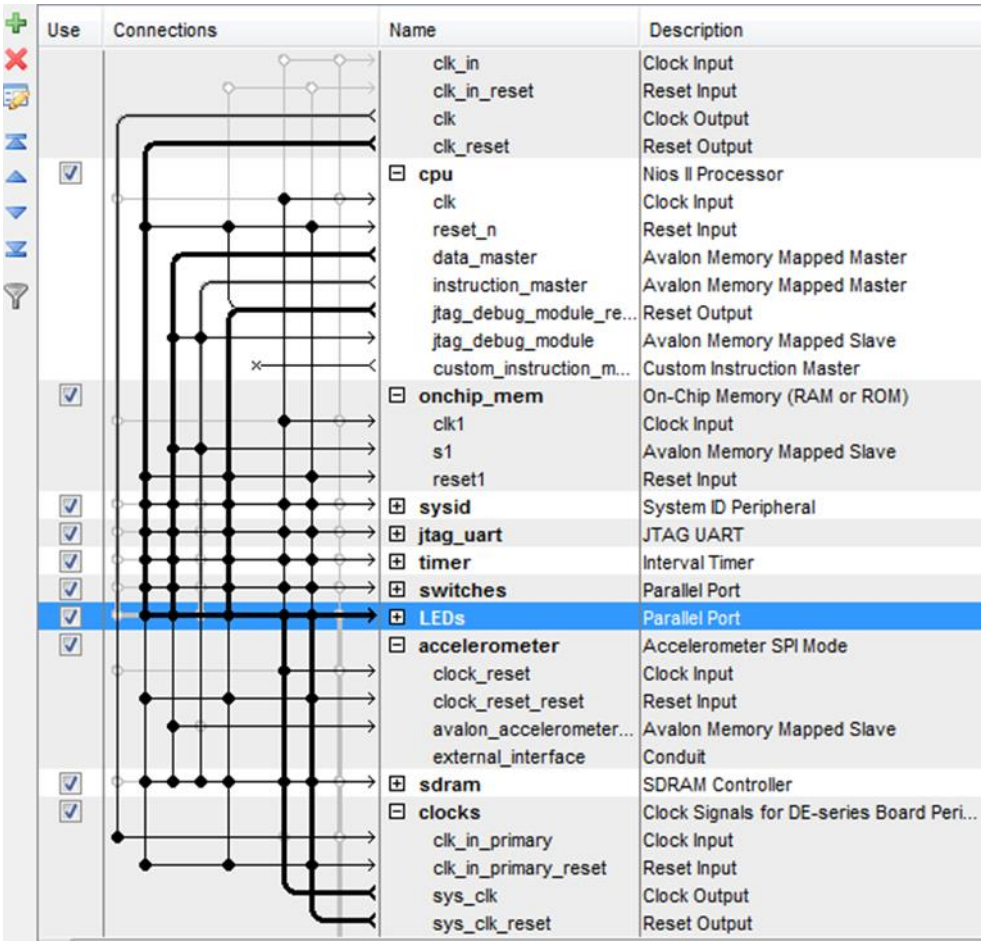


Figura 4. Conexiones entre los componentes del sistema de hardware

6. Se exportan las señales de los componentes a las que posteriormente se le realizarán asignación de pines de la FPGA para ello en la columna **Export** se da doble clic sobre las señales a exportar y se cambia los nombres como se muestra en la *Figura 5*.

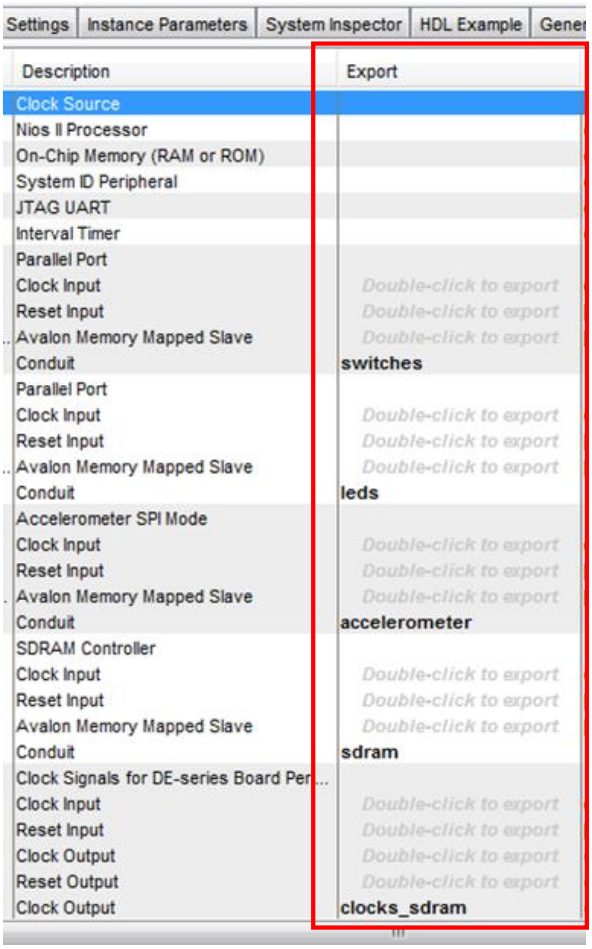


Figura 5. Señales exportadas del sistema

7. Se debe añadir las solicitudes de interrupción o IRQs (Interrupt Request). En la columna IRQ, conectar el Procesador Nios II al JTAG UART, al Interval Timer y al controlador del acelerómetro. Clic en el valor IRQ del jtag_uart y escribir 8, de manera similar escribir 0 para el valor IRQ del componente sys_clk_timer y 15 para el componente Accelerometer SPI Mode.

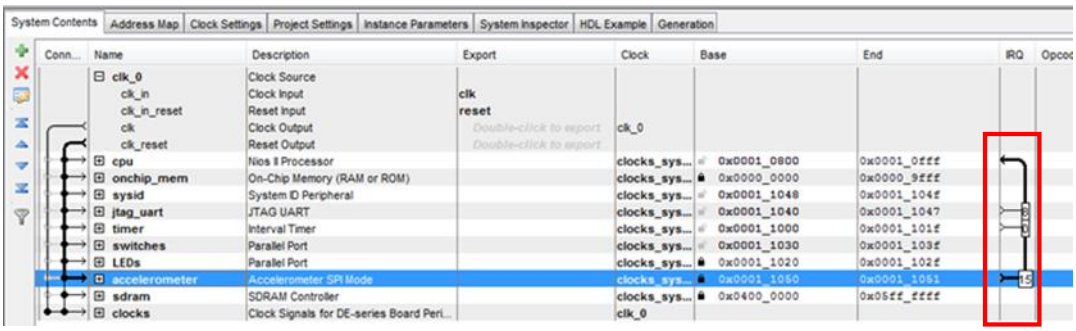


Figura 6. Asignación de la prioridad de interrupciones

8. Añadir las direcciones base, como sugerencia establecer manualmente la dirección base de la on-chip memory a 0x00000000 y fijarlo a ese valor seleccionando el candado ubicado a lado izquierdo de la dirección, para todos los demás componentes generar las direcciones de forma automática mediante la opción *Assign Base Addresses* en la pestaña System.

9. Guardar, llenar el nombre de archivo (sugerido system_accelerometer). Clic en la pestaña **Generation** y seleccionar None tanto en la sección **Simulation** como en **Testbench System.** Clic **Generate,** al finalizar clic en **Close,** cerrar Qsys y regresar a Quartus II.

10. Se procede a añadir el archivo de extensión .qip generado en Qsys, que contiene la información del sistema Nios II creado. En la ventana de Project Navigator, dar doble clic sobre la carpeta Files, y buscar el archivo que se encuentra en la ruta: <nombre deldirectorio>/system_accelerometer/synthesis/system_accelerometer.qip(Verificar que se esté mostrando todos los archivos o los de extensión .qip al buscar).

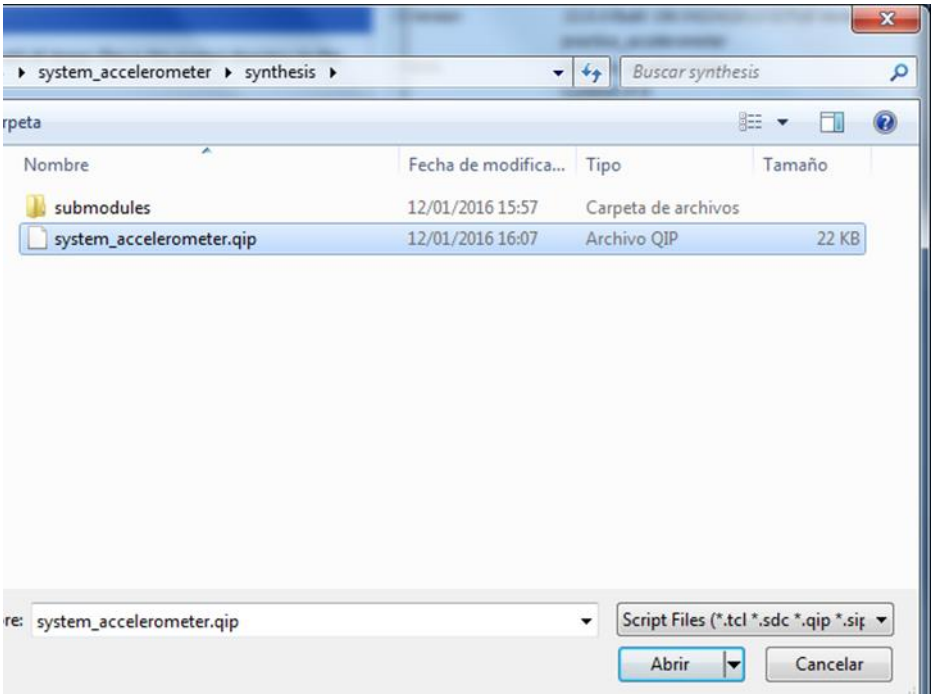


Figura 7. Adición del archivo .qip al proyecto en quartus

11. Se procede a instanciar el módulo del sistema creado en Qsys, para ello crear un nuevo *Block Diagram / Schematic File*, y añadir el bloque system_accelerometer. Agregar los terminales de entrada y salida, con sus respectivas etiquetas. Guardar el archivo (practica_accelerometer).

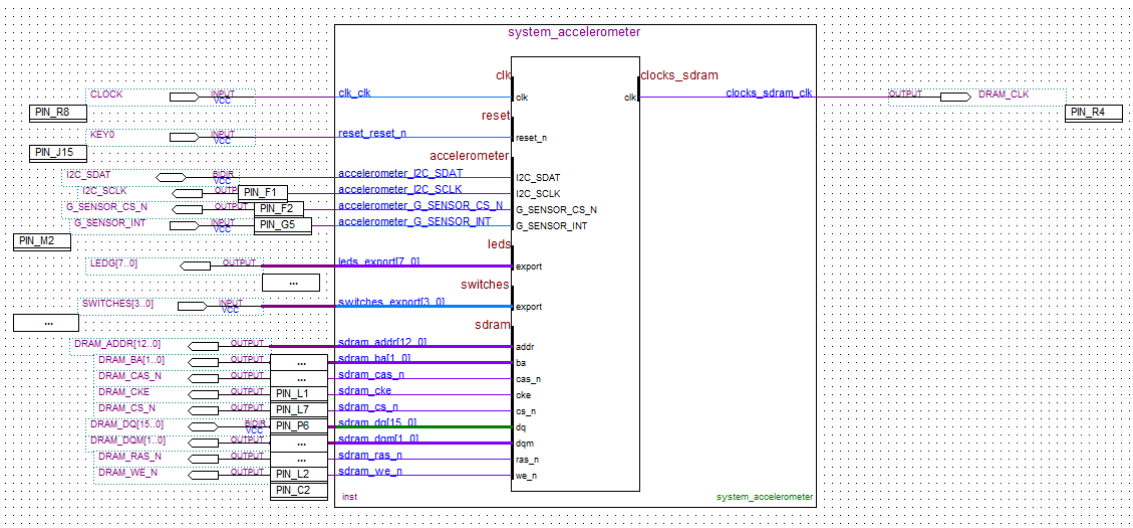


Figura 8. Adición de los pines al nuevo bloque adc_system

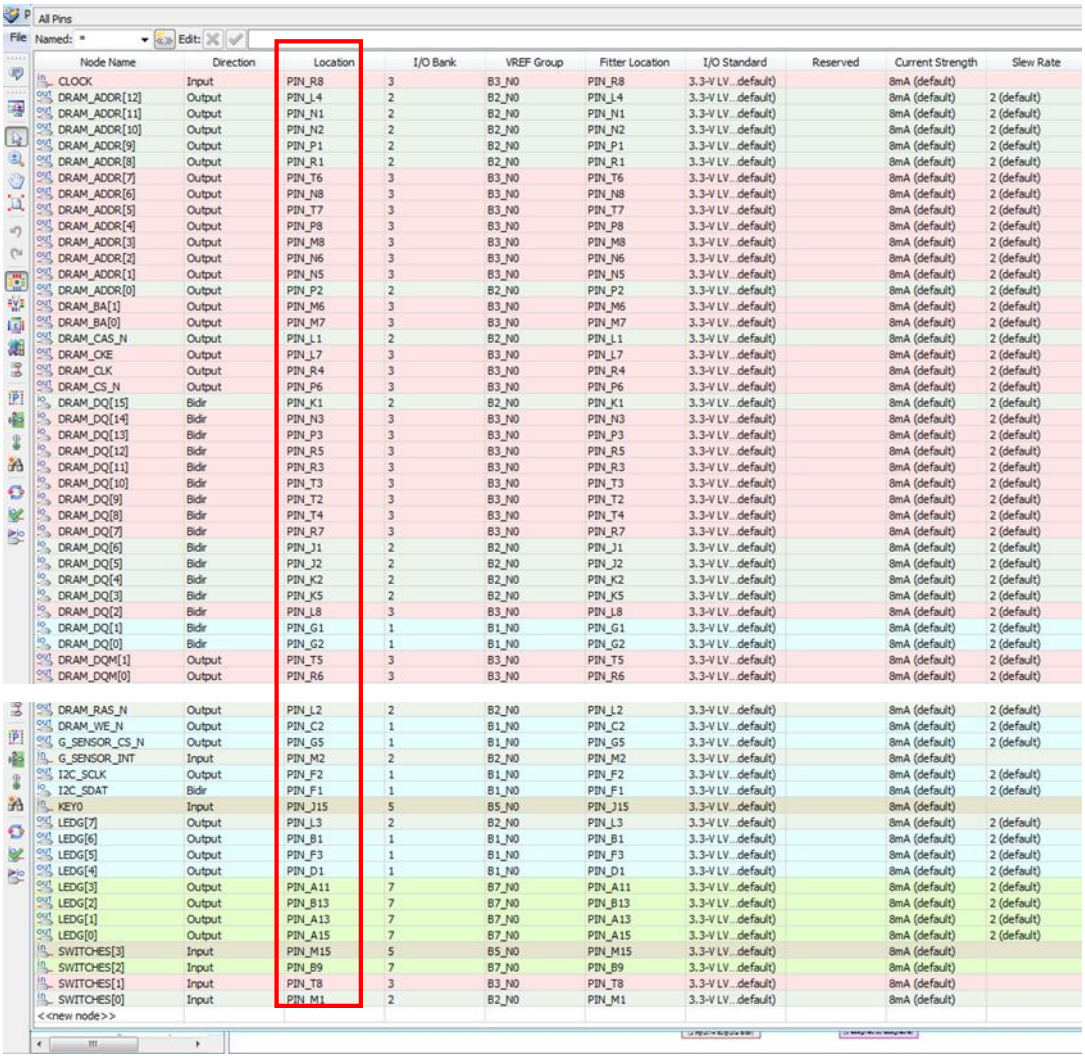
12. En el menú **Processing** seleccionar **Start** > **Start Analysis & Elaboration,** con lo cual al revisar el **Pin Planner** observamos listadas las entradas y salidas creadas.

13. Para la asignación de pines nos referimos al manual del usuario de la tarjeta DE0-Nano y encontramos que se debe realizar la siguiente asignación (Standard I/O 3.3LVTTL), para emplear los switches , leds, clock y acelerómetro de la tarjeta.

    En el directorio del proyecto existe un archivo con la extensión .qsf, el mismo que contiene la asignación de pines, en caso de tener en digital la asignación de pines puede añadirse a la parte ya existente. Se puede copiar el siguiente código en el archivo .qsf ubicado en el directorio similar a la práctica 2. Para una operación apropiada debe cambiarse el estándar, en donde dice “2.5 V” cambiar por “3.3-V LVTTL”. Guarde los cambios del archivo y ciérrelo.

```
set_global_assignment -name STRATIX_DEVICE_IO_STANDARD "3.3-V LVTTL"
set_location_assignment PIN_R8 -to CLOCK
set_location_assignment PIN_G5 -to G_SENSOR_CS_N
set_location_assignment PIN_M2 -to G_SENSOR_INT
set_location_assignment PIN_F2 -to I2C_SCLK
set_location_assignment PIN_F1 -to I2C_SDAT
set_location_assignment PIN_J15 -to KEY0
set_location_assignment PIN_L3 -to LEDG[7]
set_location_assignment PIN_B1 -to LEDG[6]
set_location_assignment PIN_F3 -to LEDG[5]
set_location_assignment PIN_D1 -to LEDG[4]
set_location_assignment PIN_A11 -to LEDG[3]
set_location_assignment PIN_B13 -to LEDG[2]
set_location_assignment PIN_A13 -to LEDG[1]
set_location_assignment PIN_A15 -to LEDG[0]
set_location_assignment PIN_M15 -to SWITCHES[3]
set_location_assignment PIN_B9 -to SWITCHES[2]
set_location_assignment PIN_T8 -to SWITCHES[1]
set_location_assignment PIN_M1 -to SWITCHES[0]
set_location_assignment PIN_L4 -to DRAM_ADDR[12]
set_location_assignment PIN_N1 -to DRAM_ADDR[11]
set_location_assignment PIN_N2 -to DRAM_ADDR[10]
set_location_assignment PIN_P1 -to DRAM_ADDR[9]
set_location_assignment PIN_R1 -to DRAM_ADDR[8]
set_location_assignment PIN_T6 -to DRAM_ADDR[7]
set_location_assignment PIN_N8 -to DRAM_ADDR[6]
set_location_assignment PIN_T7 -to DRAM_ADDR[5]
```

```
set_location_assignment PIN_P8 -to DRAM_ADDR[4]
set_location_assignment PIN_M8 -to DRAM_ADDR[3]
set_location_assignment PIN_N6 -to DRAM_ADDR[2]
set_location_assignment PIN_N5 -to DRAM_ADDR[1]
set_location_assignment PIN_P2 -to DRAM_ADDR[0]
set_location_assignment PIN_M6 -to DRAM_BA[1]
set_location_assignment PIN_M7 -to DRAM_BA[0]
set_location_assignment PIN_L1 -to DRAM_CAS_N
set_location_assignment PIN_L7 -to DRAM_CKE
set_location_assignment PIN_R4 -to DRAM_CLK
set_location_assignment PIN_P6 -to DRAM_CS_N
set_location_assignment PIN_K1 -to DRAM_DQ[15]
set_location_assignment PIN_N3 -to DRAM_DQ[14]
set_location_assignment PIN_P3 -to DRAM_DQ[13]
set_location_assignment PIN_R5 -to DRAM_DQ[12]
set_location_assignment PIN_R3 -to DRAM_DQ[11]
set_location_assignment PIN_T3 -to DRAM_DQ[10]
set_location_assignment PIN_T2 -to DRAM_DQ[9]
set_location_assignment PIN_T4 -to DRAM_DQ[8]
set_location_assignment PIN_R7 -to DRAM_DQ[7]
set_location_assignment PIN_J1 -to DRAM_DQ[6]
set_location_assignment PIN_J2 -to DRAM_DQ[5]
set_location_assignment PIN_K2 -to DRAM_DQ[4]
set_location_assignment PIN_K5 -to DRAM_DQ[3]
set_location_assignment PIN_L8 -to DRAM_DQ[2]
set_location_assignment PIN_G1 -to DRAM_DQ[1]
set_location_assignment PIN_G2 -to DRAM_DQ[0]
set_location_assignment PIN_L2 -to DRAM_RAS_N
set_location_assignment PIN_C2 -to DRAM_WE_N
set_location_assignment PIN_R6 -to DRAM_DQM[0]
set_location_assignment PIN_T5 -to DRAM_DQM[1]
```



All Pins

Named: * Edit:

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate
CLOCK	Input	PIN_R8	3	B3_N0	PIN_R8	3.3-V LV...default)		8mA (default)	
DRAM_ADDR[12]	Output	PIN_L4	2	B2_N0	PIN_L4	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_ADDR[11]	Output	PIN_N1	2	B2_N0	PIN_N1	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_ADDR[10]	Output	PIN_N2	2	B2_N0	PIN_N2	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_ADDR[9]	Output	PIN_P1	2	B2_N0	PIN_P1	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_ADDR[8]	Output	PIN_R1	2	B2_N0	PIN_R1	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_ADDR[7]	Output	PIN_T6	3	B3_N0	PIN_T6	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_ADDR[6]	Output	PIN_N8	3	B3_N0	PIN_N8	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_ADDR[5]	Output	PIN_T7	3	B3_N0	PIN_T7	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_ADDR[4]	Output	PIN_P8	3	B3_N0	PIN_P8	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_ADDR[3]	Output	PIN_M8	3	B3_N0	PIN_M8	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_ADDR[2]	Output	PIN_N6	3	B3_N0	PIN_N6	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_ADDR[1]	Output	PIN_N5	3	B3_N0	PIN_N5	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_ADDR[0]	Output	PIN_P2	2	B2_N0	PIN_P2	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_BA[1]	Output	PIN_M6	3	B3_N0	PIN_M6	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_BA[0]	Output	PIN_M7	3	B3_N0	PIN_M7	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_CAS_N	Output	PIN_L1	2	B2_N0	PIN_L1	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_CKE	Output	PIN_L7	3	B3_N0	PIN_L7	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_CLK	Output	PIN_R4	3	B3_N0	PIN_R4	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_CS_N	Output	PIN_P6	3	B3_N0	PIN_P6	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_DQ[15]	Bidir	PIN_K1	2	B2_N0	PIN_K1	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_DQ[14]	Bidir	PIN_N3	3	B3_N0	PIN_N3	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_DQ[13]	Bidir	PIN_P3	3	B3_N0	PIN_P3	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_DQ[12]	Bidir	PIN_R5	3	B3_N0	PIN_R5	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_DQ[11]	Bidir	PIN_R3	3	B3_N0	PIN_R3	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_DQ[10]	Bidir	PIN_T3	3	B3_N0	PIN_T3	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_DQ[9]	Bidir	PIN_T2	3	B3_N0	PIN_T2	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_DQ[8]	Bidir	PIN_T4	3	B3_N0	PIN_T4	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_DQ[7]	Bidir	PIN_R7	3	B3_N0	PIN_R7	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_DQ[6]	Bidir	PIN_J1	2	B2_N0	PIN_J1	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_DQ[5]	Bidir	PIN_J2	2	B2_N0	PIN_J2	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_DQ[4]	Bidir	PIN_K2	2	B2_N0	PIN_K2	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_DQ[3]	Bidir	PIN_K5	2	B2_N0	PIN_K5	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_DQ[2]	Bidir	PIN_L8	3	B3_N0	PIN_L8	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_DQ[1]	Bidir	PIN_G1	1	B1_N0	PIN_G1	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_DQ[0]	Bidir	PIN_G2	1	B1_N0	PIN_G2	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_DQM[1]	Output	PIN_T5	3	B3_N0	PIN_T5	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_DQM[0]	Output	PIN_R6	3	B3_N0	PIN_R6	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_RAS_N	Output	PIN_L2	2	B2_N0	PIN_L2	3.3-V LV...default)		8mA (default)	2 (default)
DRAM_WE_N	Output	PIN_C2	1	B1_N0	PIN_C2	3.3-V LV...default)		8mA (default)	2 (default)
G_SENSOR_CS_N	Output	PIN_G5	1	B1_N0	PIN_G5	3.3-V LV...default)		8mA (default)	2 (default)
G_SENSOR_INT	Input	PIN_M2	2	B2_N0	PIN_M2	3.3-V LV...default)		8mA (default)	
I2C_SCLK	Output	PIN_F2	1	B1_N0	PIN_F2	3.3-V LV...default)		8mA (default)	2 (default)
I2C_SDAT	Bidir	PIN_F1	1	B1_N0	PIN_F1	3.3-V LV...default)		8mA (default)	2 (default)
KEY0	Input	PIN_J15	5	B5_N0	PIN_J15	3.3-V LV...default)		8mA (default)	
LEDG[7]	Output	PIN_L3	2	B2_N0	PIN_L3	3.3-V LV...default)		8mA (default)	2 (default)
LEDG[6]	Output	PIN_B1	1	B1_N0	PIN_B1	3.3-V LV...default)		8mA (default)	2 (default)
LEDG[5]	Output	PIN_F3	1	B1_N0	PIN_F3	3.3-V LV...default)		8mA (default)	2 (default)
LEDG[4]	Output	PIN_D1	1	B1_N0	PIN_D1	3.3-V LV...default)		8mA (default)	2 (default)
LEDG[3]	Output	PIN_A11	7	B7_N0	PIN_A11	3.3-V LV...default)		8mA (default)	2 (default)
LEDG[2]	Output	PIN_B13	7	B7_N0	PIN_B13	3.3-V LV...default)		8mA (default)	2 (default)
LEDG[1]	Output	PIN_A13	7	B7_N0	PIN_A13	3.3-V LV...default)		8mA (default)	2 (default)
LEDG[0]	Output	PIN_A15	7	B7_N0	PIN_A15	3.3-V LV...default)		8mA (default)	2 (default)
SWITCHES[3]	Input	PIN_M15	5	B5_N0	PIN_M15	3.3-V LV...default)		8mA (default)	
SWITCHES[2]	Input	PIN_B9	7	B7_N0	PIN_B9	3.3-V LV...default)		8mA (default)	
SWITCHES[1]	Input	PIN_T8	3	B3_N0	PIN_T8	3.3-V LV...default)		8mA (default)	
SWITCHES[0]	Input	PIN_M1	2	B2_N0	PIN_M1	3.3-V LV...default)		8mA (default)	
<<new node>>									

Figura 9. Pines asignados a cada señal exportada

14. Compilar el proyecto para crear el archivo .sof que se descargará en la tarjeta, luego de lo cual podemos observar el reporte de compilación que fue exitoso y la utilización de elementos lógicos de la FPGA, también muestra el uso de la memoria contenida en la FPGA (On-chip Memory), que está al 62%, y la misma en esta práctica únicamente maneja datos temporales y provee un control del flujo del sistema; pero no se la utiliza como memoria del programa ni de datos del programa a compilar sobre el hardware diseñado.

    Otro dato relevante es el uso de un PLL que se genera al utilizar el componente *Clock Signals for DE-series Board Peripherals* de Qsys para

satisfacer los requerimientos de la SDRAM de adelantar al reloj del procesador por 3ns.

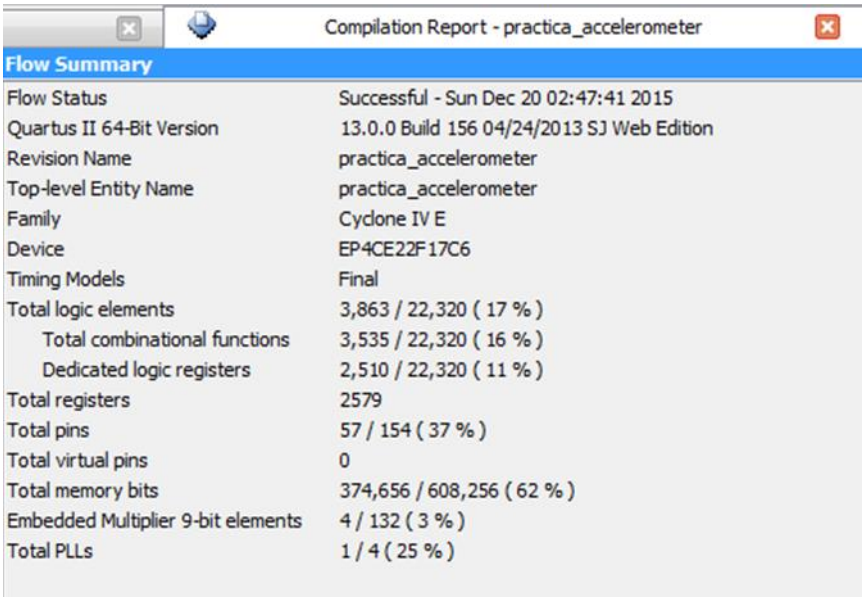


Figura 10. Reporte de compilación del sistema de hardware

15. Descargar el archivo .sof en la tarjeta DE0-Nano con la herramienta de programación (**Tools > Programmer**).

16. Abrir el programa Nios II SBT desde quartus en la pestaña de herramientas (**Tools > Nios II Software Build Tools for Eclipse**) y se procede a la creación del software a correr sobre el hardware diseñado previamente, para ello crear una carpeta en el directorio del proyecto con el nombre workspace (sugerido). Crear un proyecto en blanco (nombre sugerido: accelerometer), seleccionando el correspondiente archivo .sopcinfo. Al proyecto creado añadir un archivo main.c, en el mismo se escribirá el siguiente código:

```
#include <stdio.h>

int main(void){
```

```
        volatile char * Accelerometer = (volatile char *) 0x00011050;
        volatile int * Green_LEDs = (volatile int *) 0x00011020;

        int temp;
        int x_axis;

        while(1){
              usleep(1000000);
        //Select and read the lower-order byte of the x-axis value
              *(Accelerometer) = 0x32;
              x_axis = 0x000000FF & *(Accelerometer+1);
        //Select and read the higher-order byte of the x-axis value
              *(Accelerometer) = 0x33;
              temp = 0xFF & *(Accelerometer+1);
        //Combine the two x-axis bytes, and sign-extend it
              x_axis += temp << 8;
              if (x_axis & 0x008000)
                    x_axis |= 0xFFFF0000;
        //Write the x-axis value out to the LEDs
              *(Green_LEDs) = x_axis;
              printf("Valor medido es:->  %d \n  ",x_axis);
        }
        return 0;
}
```

En el código mostrado debe tenerse en cuenta que correspondan las direcciones de los punteros tanto a los leds como al acelerómetro, esto se puede ver en Qsys en la columna de direcciones base. Además también se encuentra en la carpeta del proyecto accelerometer_bsp en el archivo system.h.

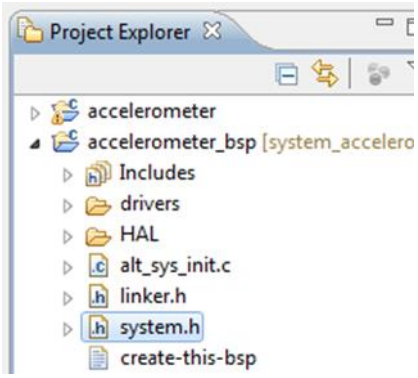


Figura 11. Localización del archivo system.h

Al abrir el archivo system.h se tienen las características del hardware, y se busca la parte del código que contiene la información de los leds y del acelerómetro, donde se visualizan sus direcciones base.

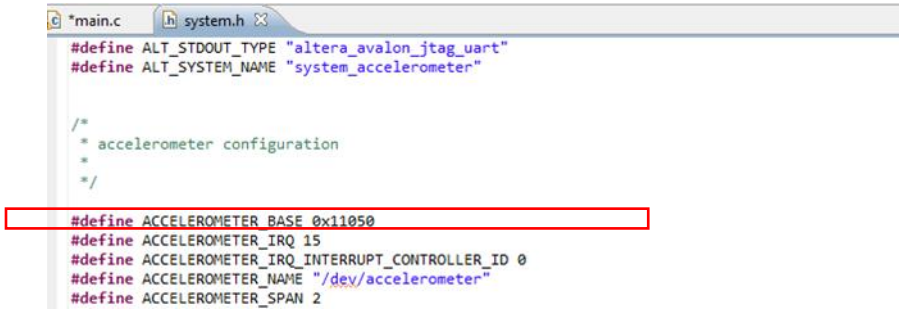


Figura 12. Verificación de las direcciones del componente acelerómetro

Podemos observar también en el código que al apuntar al registro 0x32 se da lectura al byte menos significativo del eje X y al apuntar al registro 0x33 se da lectura al byte más significativo del eje X, en el caso que se quisiera obtener los datos de los ejes Y y Z se deben apuntar a los registros 0x34, 0x35, 0x36 y 0x37.

17. Clic derecho sobre el proyecto *accelerometer* y seleccionar **Build Project**, al finalizar se muestra un mensaje de finalizado. Nuevamente, clic derecho sobre el proyecto *contador* y seleccionar **Run As** > **Nios II Hardware.**

Al correr el programa podemos visualizar en consola y en los Leds de la tarjeta el valor de los 8 bits menos significativos de la lectura del eje X. Por ejemplo si la lectura del acelerómetro sería 196 este valor decimal corresponde a 0.767g en valor gravitacional, esta conversión se describe en el correspondiente anexo de la presente práctica.

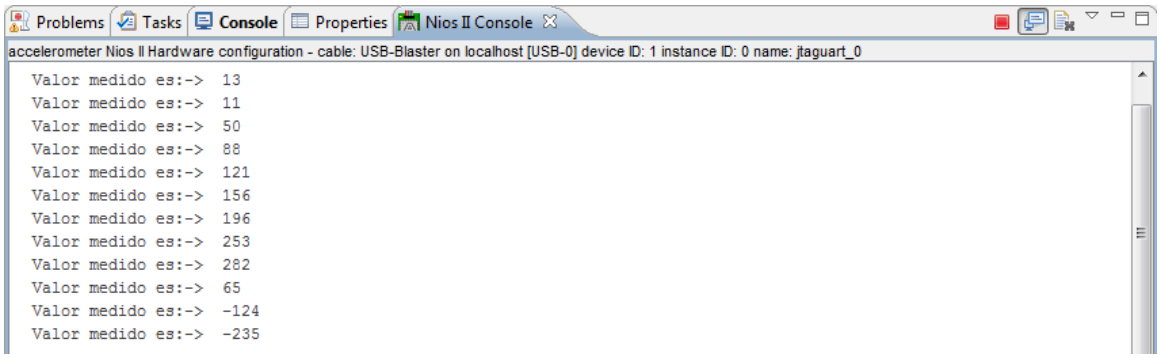


Figura 13. Consola de Nios II SBT mostrando la lectura de inclinación del eje X

PRÁCTICA # 5

TEMA: *Sistemas embebidos de arranque mediante el uso de la memoria flash Spansion S25FL064P, integrada en la tarjeta DE0-Nano y uso del Altera Monitor Program.*

1. Objetivos:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Entender y explicar el uso de memorias no volátiles en el desarrollo de sistemas embebidos.
- Ampliar la destreza y entendimiento sobre los componentes empleados para el desarrollo del hardware del sistema Nios II con la herramienta Qsys.
- Escribir un programa de arranque sobre el hardware desarrollado utilizando *'Nios II Software Build for Eclipse'.*
- Emplear la aplicación de software *Monitor Program* como método alternativo para el desarrollo de sistemas embebidos en tarjetas de Altera.

2. Requisitos mínimos:

- Tener instalado Quartus II y Nios II EDS (versión 13.0).
- Conocimientos básicos de Quartus II y programación en VHDL y C.
- Haber culminado satisfactoriamente la Práctica # 2.
- Lectura y comprensión del archivo: '*Anexo_A.5_Practica_5_Fundamentación Teórica*'

3. Breve explicación de la práctica:

Se desarrolla un sistema de hardware que incluye un procesador embebido de Altera Nios II en conjunto con otros módulos *véase Figura 1)*. Todos los componentes se conectan por medio de una interconexión llamada *Avalon switch fabric.* De esta manera se forma un sistema que luego se implementa en el chip FPGA utilizando el programa de *Quartus II*.

Todas las partes del sistema basado en el procesador Nios II, son definidas usando lenguaje de descripción de hardware utilizando la herramienta *Qsys* para implementar el sistema deseado simplemente seleccionando los componentes requeridos y especificando los parámetros necesarios. Como memoria del programa y de datos se utiliza la memoria contenida en la misma FPGA, debido que satisface la capacidad que se requiere.

Sobre el diseño de hardware desarrollado se corre un programa que permite mostrar en los leds de la tarjeta un desplazamiento de izquierda a derecha y viceversa, primero se lo realiza con *Nios II Software Build Tools for Eclipse* y luego con Altera Monitor Program. Finalmente esta básica aplicación de software se la convierte en un programa de arranque, utilizando la memoria flash *Spansion S25FL064P* contenida en la tarjeta Deo-Nano.

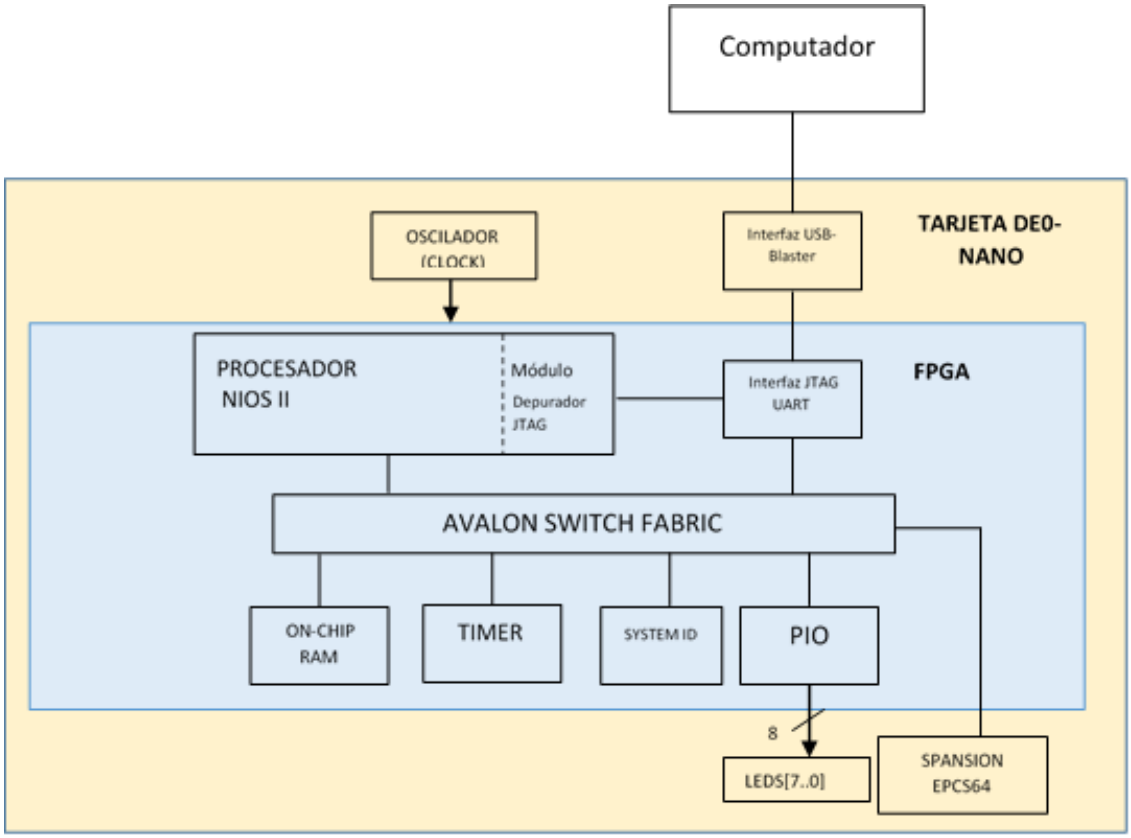


Figura 1. Diagrama de bloques del sistema de hardware a desarrollar en la Práctica # 5

4. Desarrollo de la práctica:

1. Abrir el programa Quartus II y crear un proyecto nuevo (*Nombre del proyecto sugerido: practica_flash_memory)* con la familia *Cyclone IV E, dispositivo EP4CE22F17C6* y guardarlo en un nuevo directorio (practica_5) donde se almacenarán todos los archivos que se generen en el desarrollo de la práctica. ***Nota****. En la ventana EDA tool settings, seleccione <None> en todas las opciones de la columna Tool Name.*

2. Una vez creado el proyecto en Quartus, seleccionar **Tools > Qsys**.

3. Añadir los componentes básicos del sistema desde la librería, para ello seleccionar y agregar los que se describen en la *Tabla 1* con sus respectivas configuraciones:

Componente	**Renombrar componentes**	**Configuración**
Nios II Processor	cpu	Nios II/e
On-Chip Memory (RAM or ROM)	onchip_memory_SRAM	**
System ID Peripherial	sysid	default
JTAG UART	default	default
Interval Timer	timer	default
Parallel Port (University Program)	Switches	DE-Series Borad : DE0-Nano I/O device: Dip Switches
Parallel Port (University Program)	LEDs	DE-Series Board : DE0-Nano I/O device: LEDs

** Las configuraciones se detallan más adelante

Tabla 1. Componentes básicos del sistema de hardware de la práctica # 5

En caso de requerir procedimientos más detallados para agregar los componentes antes mencionados refiérase a la práctica 1 que contiene parte introductoria al uso de Qsys para el desarrollo de sistemas embebidos con el procesador Nios II.

4. El componente onchip_memory_SRAM se configura para que al arrancar el sistema sobre este se copie el sistema que se almacenará en la memoria flash. Dar doble clic sobre el componente y ajustar las siguientes configuraciones:
    - Marcar la casilla Dual-port access.- Esta es un variación del On-chip Memory que permite trabajar con puertos separados para las

transacciones de lectura y escritura, permitiendo de esta manera duplicar la capacidad de ancho de banda de la memoria y así ser escrita y leída simultáneamente por diferentes puertos.

- Total memory size: 30KB (30720 bytes)
- Marcar la casilla de *Initialize memory content.- Debido a que el procesador Nios II ejecutará las instrucciones en el onchip_memory_SRAM los contenidos deben ser inicializados durante la configuración de la FPGA.*
- Marcar la casilla de *Enable non-default initialization file* y en el espacio de *User created initialization file* escribir el nombre **boot_code.hex**.- Este archivo de inicialización se creará posterioremente.

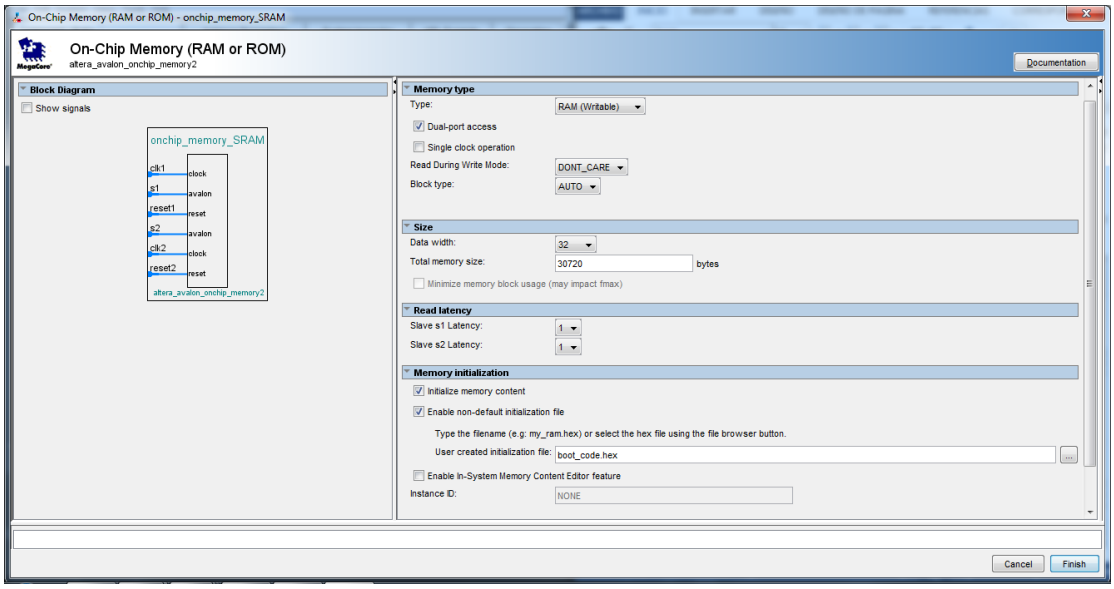


Figura 2. Ajuste de los parámetros del componente onchip_memory_SRAM

5. Recordar que los errores que se visualizan se deben a la falta de conexiones, de direcciones base a los componentes y asignaciones de vectores de memoria de reset y de excepción en el procesador. Las conexiones a realizar se muestran en la *Figura 3*.

6. El puerto s1 del componente onchip_memory_SRAM conectarlo al *instruction_master* del cpu y el s2 al data_master del cpu como se muestra en la *Figura 3*. Se exportan las señales de los componentes a las que posteriormente se le realizarán asignación de pines de la FPGA para ello en la columna **Export** se da doble clic sobre las señales a exportar y se cambia los nombres como se muestra en la *Figura 3*.

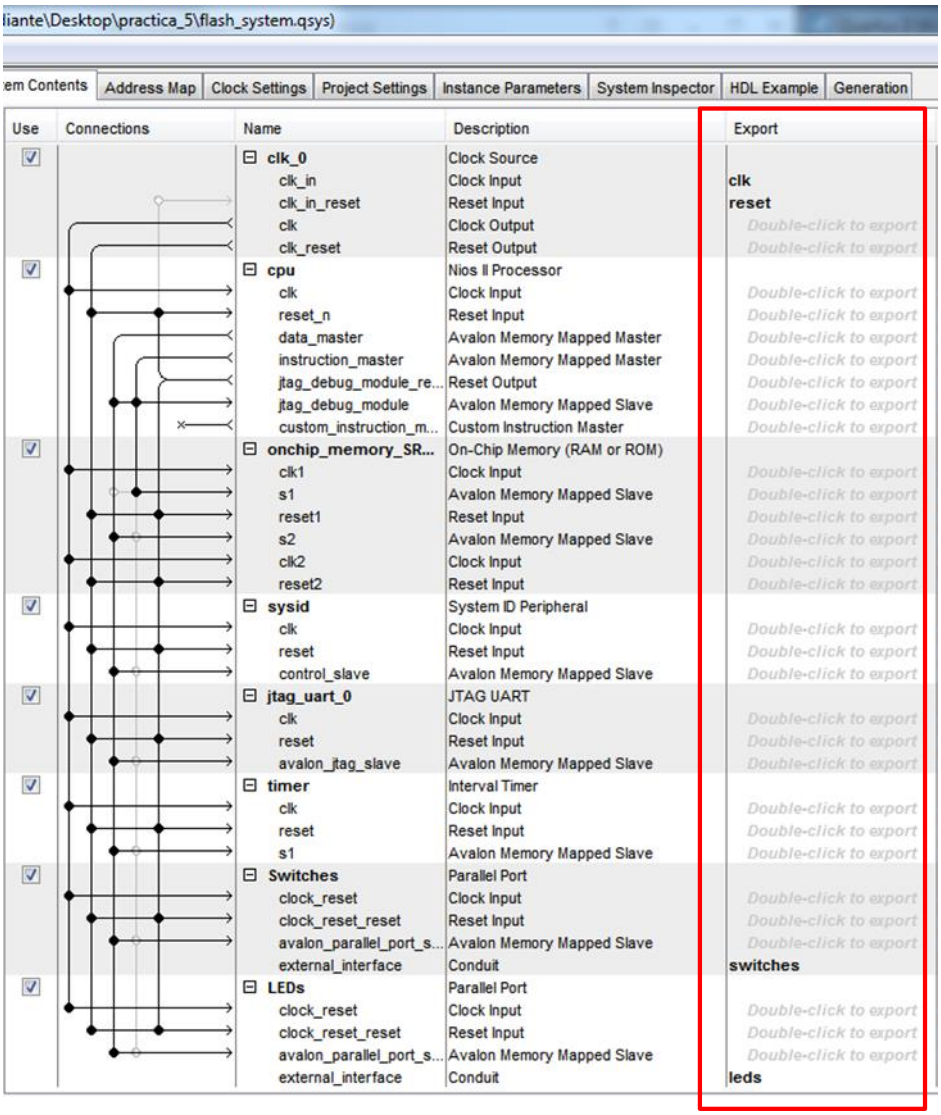


Figura 3. Conexiones entre los componentes del sistema de hardware y señales exportadas

7. En las configuraciones del componente cpu, seleccionar el onchip_memory_SRAM.s1 como el reset vector y el exception vector.

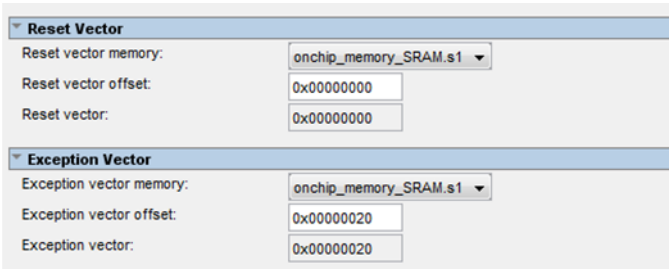


Figura 4. Asignación de los vectores reset y exception

8. Se debe añadir las solicitudes de interrupción o IRQs (Interrupt Request). En la columna IRQ, conectar el Procesador Nios II al JTAG UART y al Interval Timer. Clic en el valor IRQ del jtag_uart y escribir 8, de manera similar escribir 0 para el valor IRQ del componente sys_clk_timer

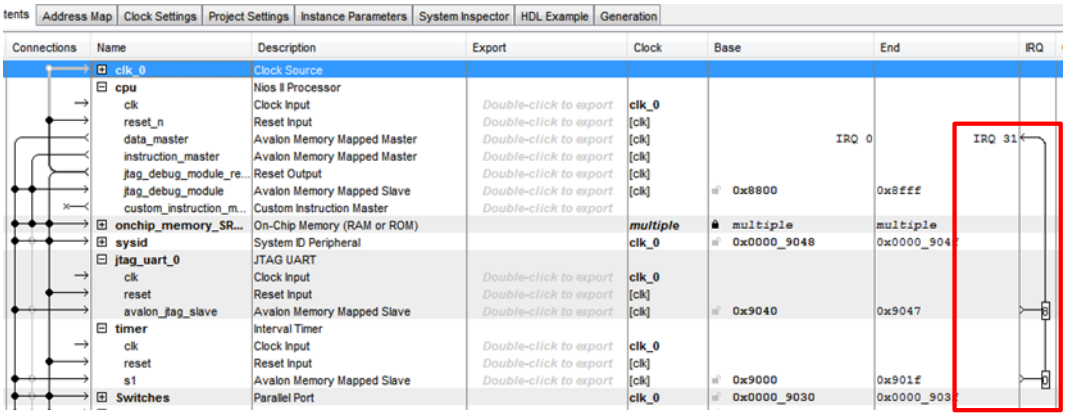


Figura 5. Asignación de la prioridad de interrupciones

9. Añadir las direcciones base, como sugerencia establecer manualmente la dirección base de la on-chip memory a 0x00000000 y fijarlo a ese valor seleccionando el candado ubicado a lado izquierdo de la dirección, para todos los demás componentes generar las direcciones de forma automática mediante la opción *Assign Base Addresses* en la pestaña System.

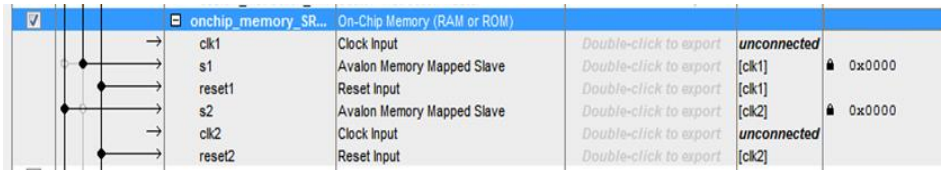


Figura 6. Asignación de dirección base al componente onchip_memory_SRAM

10. Guardar, llenar el nombre de archivo (sugerido flash_system). Clic en la pestaña **Generation** y seleccionar None tanto en la sección **Simulation** como en **Testbench System.** Clic **Generate,** al finalizar clic en **Close,** cerrar Qsys y regresar a Quartus II.

11. Se procede a añadir el archivo de extensión .qip generado en Qsys, que contiene la información del sistema Nios II creado. En la ventana de Project Navigator, dar doble clic sobre la carpeta Files, y buscar el archivo que se encuentra en la ruta: <nombre deldirectorio>/flash_system/synthesis/flash_system.qip(Verificar que se esté mostrando todos los archivos o los de extensión .qip al buscar).

12. Se procede a instanciar el módulo del sistema creado en Qsys, para ello crear un nuevo *Block Diagram / Schematic File*, y añadir el bloque flash_system. Agregar los terminales de entrada y salida, con sus respectivas etiquetas. Guardar el archivo (practica_flash_memory).

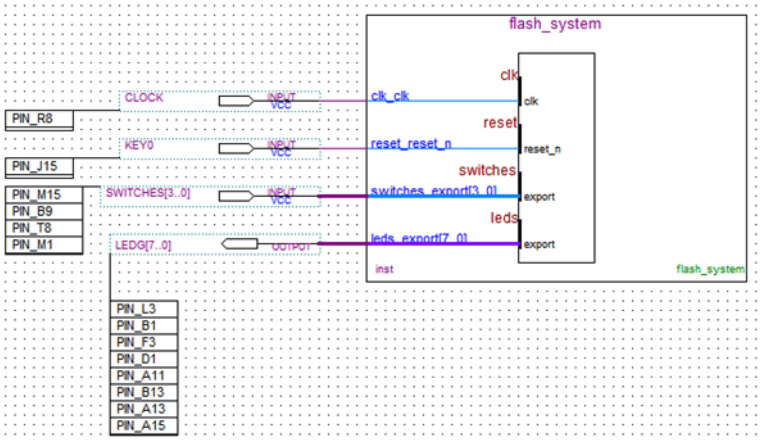


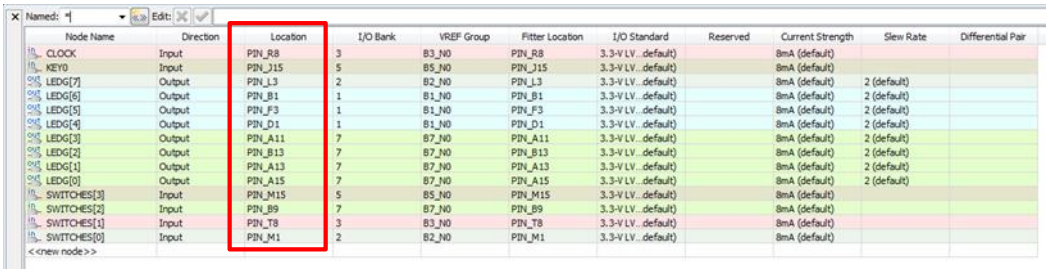
Figura 7. Adición de los pines al nuevo bloque flash_system

13. En el menú **Processing** seleccionar **Start** > **Start Analysis & Elaboration,** con lo cual al revisar el **Pin Planner** observamos listadas las entradas y salidas creadas.

14. Para la asignación de pines nos referimos al manual del usuario de la tarjeta DE0-Nano y encontramos que se debe realizar la siguiente asignación (Standard I/O 3.3LVTTL), para emplear los switches , leds, clock y pulsador de la tarjeta.

    En el directorio del proyecto existe un archivo con la extensión .qsf, el mismo que contiene la asignación de pines, en caso de tener en digital la asignación de pines puede añadirse a la parte ya existente. Se puede copiar el siguiente código en el archivo .qsf ubicado en el directorio similar a las prácticas previas. Para una operación apropiada debe cambiarse el estándar, en donde dice “2.5 V” cambiar por “3.3-V LVTTL”. Guarde los cambios del archivo y ciérrelo.

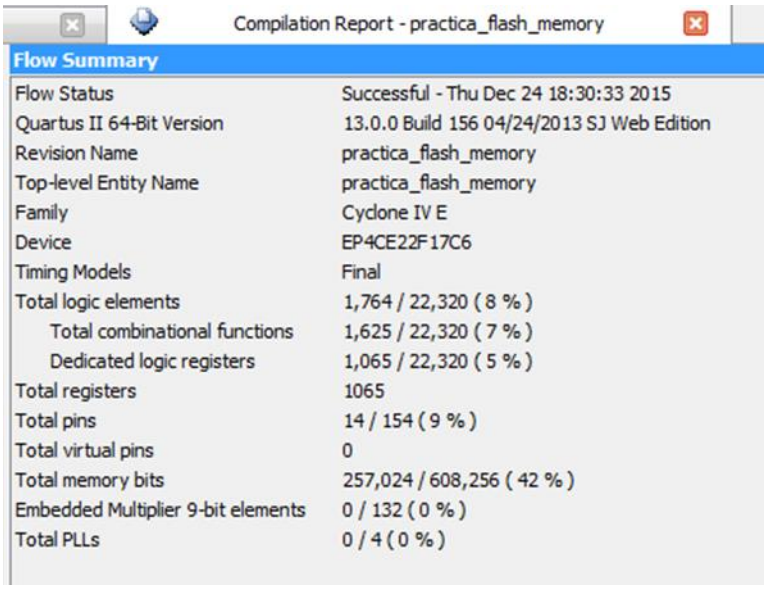
```
set_global_assignment -name STRATIX_DEVICE_IO_STANDARD "3.3-V LVTTL"
set_location_assignment PIN_R8 -to CLOCK
set_location_assignment PIN_J15 -to KEY0
set_location_assignment PIN_L3 -to LEDG[7]
set_location_assignment PIN_B1 -to LEDG[6]
set_location_assignment PIN_F3 -to LEDG[5]
set_location_assignment PIN_D1 -to LEDG[4]
set_location_assignment PIN_A11 -to LEDG[3]
set_location_assignment PIN_B13 -to LEDG[2]
set_location_assignment PIN_A13 -to LEDG[1]
set_location_assignment PIN_A15 -to LEDG[0]
set_location_assignment PIN_M15 -to SWITCHES[3]
set_location_assignment PIN_B9 -to SWITCHES[2]
set_location_assignment PIN_T8 -to SWITCHES[1]
set_location_assignment PIN_M1 -to SWITCHES[0]
```



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair
CLOCK	Input	PIN_R8	3	B3_N0	PIN_R8	3.3-V LV...default)		8mA (default)		
KEY0	Input	PIN_J15	5	B5_N0	PIN_J15	3.3-V LV...default)		8mA (default)		
LEDG[7]	Output	PIN_L3	2	B2_N0	PIN_L3	3.3-V LV...default)		8mA (default)	2 (default)	
LEDG[6]	Output	PIN_B1	1	B1_N0	PIN_B1	3.3-V LV...default)		8mA (default)	2 (default)	
LEDG[5]	Output	PIN_F3	1	B1_N0	PIN_F3	3.3-V LV...default)		8mA (default)	2 (default)	
LEDG[4]	Output	PIN_D1	1	B1_N0	PIN_D1	3.3-V LV...default)		8mA (default)	2 (default)	
LEDG[3]	Output	PIN_A11	7	B7_N0	PIN_A11	3.3-V LV...default)		8mA (default)	2 (default)	
LEDG[2]	Output	PIN_B13	7	B7_N0	PIN_B13	3.3-V LV...default)		8mA (default)	2 (default)	
LEDG[1]	Output	PIN_A13	7	B7_N0	PIN_A13	3.3-V LV...default)		8mA (default)	2 (default)	
LEDG[0]	Output	PIN_A15	7	B7_N0	PIN_A15	3.3-V LV...default)		8mA (default)	2 (default)	
SWITCHES[3]	Input	PIN_M15	5	B5_N0	PIN_M15	3.3-V LV...default)		8mA (default)		
SWITCHES[2]	Input	PIN_B9	7	B7_N0	PIN_B9	3.3-V LV...default)		8mA (default)		
SWITCHES[1]	Input	PIN_T8	3	B3_N0	PIN_T8	3.3-V LV...default)		8mA (default)		
SWITCHES[0]	Input	PIN_M1	2	B2_N0	PIN_M1	3.3-V LV...default)		8mA (default)		
<<new node>>										

Figura 8. Pines asignados a cada señal exportada

15. Compilar el proyecto para crear el archivo .sof que se descargará en la tarjeta, luego de lo cual podemos observar el reporte de compilación que fue exitoso y la utilización de elementos lógicos de la FPGA, también muestra el uso de la memoria contenida en la FPGA (On-chip Memory), que está al 42%, cabe señalar que únicamente esta utilización es del hardware desarrollado.



Compilation Report - practica_flash_memory

Flow Summary

Flow Status	Successful - Thu Dec 24 18:30:33 2015
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	practica_flash_memory
Top-level Entity Name	practica_flash_memory
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	1,764 / 22,320 (8 %)
Total combinational functions	1,625 / 22,320 (7 %)
Dedicated logic registers	1,065 / 22,320 (5 %)
Total registers	1065
Total pins	14 / 154 (9 %)
Total virtual pins	0
Total memory bits	257,024 / 608,256 (42 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

Figura 9. Reporte de compilación del sistema de hardware

16. Descargar el archivo .sof en la tarjeta DE0-Nano con la herramienta de programación (**Tools > Programmer**).

17. Abrir el programa Nios II SBT desde quartus en la pestaña de herramientas (**Tools > Nios II Software Build Tools for Eclipse**) y se procede a la creación del software a correr sobre el hardware diseñado previamente, para ello crear una carpeta en el directorio del proyecto con el nombre workspace (sugerido). Crear un proyecto en blanco (nombre: flash_prueba), seleccionando el correspondiente archivo .sopcinfo. Al proyecto creado añadir un archivo boot_code.c, en el mismo se escribirá el siguiente código:

```
enum DIR {LEFT, RIGHT};

void sweep(int *, enum DIR *);

int main(void)

{

	volatile int *LEDG_ptr = (int *) 0x9020; // green LED address

	volatile int *timer_ptr = (int *) 0x9000; // interval timer
address

	int LEDG_bits = 1; // pattern for the green lights

	enum DIR shift_dir = LEFT; // pattern shifting direction

	/* set the interval timer period */

	int counter = 0x190000; // 1/(50 MHz) x 0x190000 = 33 msec

	*(timer_ptr + 2) = (counter & 0xFFFF);

	*(timer_ptr + 3) = (counter >> 16) & 0xFFFF;

	*(timer_ptr + 1) = 0x6; // START = 1, CONT = 1, ITO = 0

	while (1)

	{

		*LEDG_ptr = LEDG_bits; // write to the green lights

		sweep (&LEDG_bits, &shift_dir); // shift the pattern
left or right

		while ( (*timer_ptr & 0x1) == 0 ) // wait for timeout

			;
```

```
                *timer_ptr = 0; // reset the timeout bit
        }
}
        /* shift the pattern shown on the LEDs */
        void sweep (int *pattern, enum DIR *dir)
        {
                if (*dir == LEFT)
                        if (*pattern & 0x80)
                                *dir = RIGHT;
                        else
                                *pattern = *pattern << 1;
                else
                        if (*pattern & 0x01)
                                *dir = LEFT;
                        else
                                *pattern = *pattern >> 1;
        }
```

En el código mostrado debe tenerse en cuenta que correspondan las direcciones de los punteros tanto a los leds como al interval timer, a los cuales se hace referencia en el código, esto se puede ver en Qsys en la columna de direcciones base. Además también se encuentra en la carpeta del proyecto flash_prueba1_bsp en el archivo system.h.

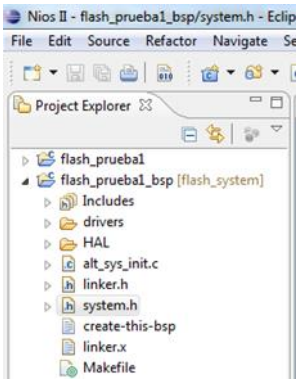


Figura 10. Localización del archivo system.h

Al abrir el archivo system.h se tienen las características del hardware, y se busca la parte del código que contiene la información de los leds y del interval timer, donde se visualizan sus direcciones base.

```
/*
 * LEDs configuration
 *
 */

#define ALT_MODULE_CLASS_LEDs altera_up_avalon_parallel_port
#define LEDS_BASE 0x9020
#define LEDS_IRQ -1
#define LEDS_IRQ_INTERRUPT_CONTROLLER_ID -1
#define LEDS_NAME "/dev/LEDs"
#define LEDS_SPAN 16
#define LEDS_TYPE "altera_up_avalon_parallel_port"
```

```
/*
 * timer configuration
 *
 */

#define ALT_MODULE_CLASS_timer altera_avalon_timer
#define TIMER_ALWAYS_RUN 0
#define TIMER_BASE 0x9000
#define TIMER_COUNTER_SIZE 32
#define TIMER_FIXED_PERIOD 0
#define TIMER_FREQ 50000000
#define TIMER_IRQ 0
#define TIMER_IRQ_INTERRUPT_CONTROLLER_ID 0
#define TIMER_LOAD_VALUE 49999
#define TIMER_MULT 0.0010
#define TIMER_NAME "/dev/timer"
#define TIMER_PERIOD 1
```

Figura 11. Verificación de las direcciones de los componentes LEDs y timer

Al correr el programa podemos visualizar en los leds de la tarjeta el desplazamiento de los leds de izquierda a derecha y viceversa. Para un entendimiento más exhaustivo del código fuente revisar el anexo de la práctica en el cual se encuentra la explicación de los registros del interval timer.

Además de *Nios II SBT for Eclipse*, otra alternativa provista por altera en su programa universitario (Altera University Program) para descargar aplicaciones de software sobre sus tarjetas de desarrollo para procesadores Nios II y ARM es el *Altera Monitor Program,* el mismo permite compilar, ensamblar, descargar y depurar programas sobre los procesadores mencionados.

Creación de un programa de arranque para el procesador Nios II y uso del Altera Monitor Program

18. Abrir el programa "*Altera Monitor Program"* (viene como parte de la instalación de *University Program), seleccionar File>New Project*.

Figura 12. Ubicación Altera Monitor Program en la computadora

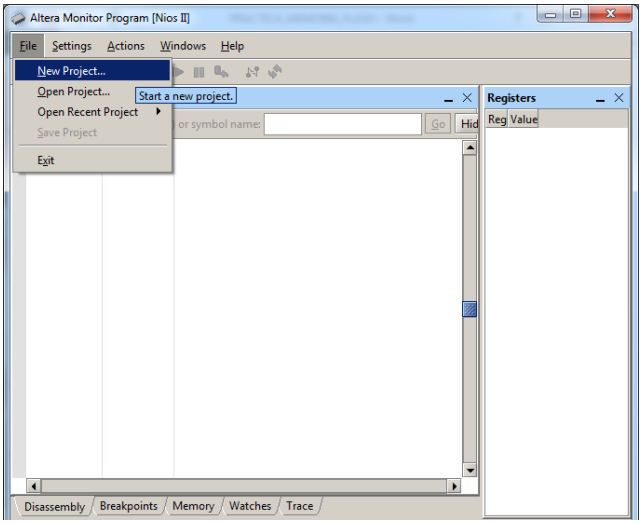


Figura 13. Interfaz Altera Monitor Program, creación de proyecto

19. En la ventana New Project Wizard, ya conocida, añadir la carpeta directorio en la que se está trabajando y el nombre del proyecto (boot_code), clic en Next. En la pantalla *Specify a system, seleccionar Custom System,* esto debido que no se utilizará ninguno de los sistemas prediseñados que provee Altera, en los detalles del sistema, añadir los archivos .sopcinfo y .sof en los espacios que corresponde a cada uno. Clic *Next.*

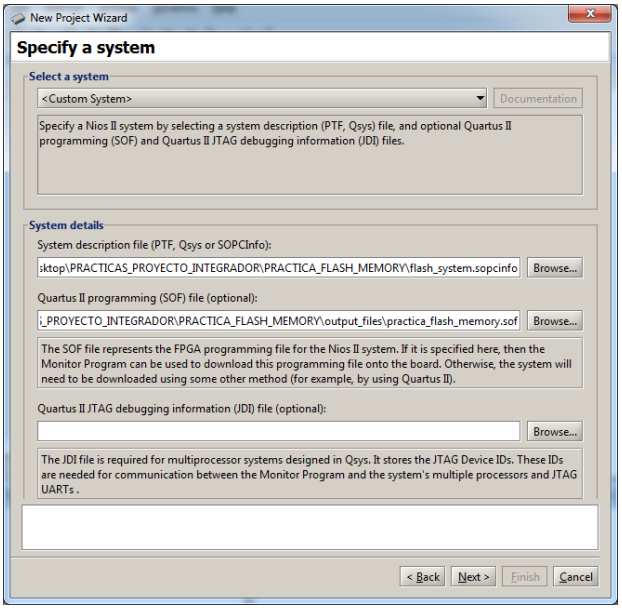


Figura 14. Especificaciones del sistema en Altera Monitor Program, creación de proyecto

20. Se debe especificar el tipo de programa con el que se cuenta para descargar a la tarjeta, seleccionar C Program. Clic *Next.*

    En la siguiente pantalla se especifican detalles del programa, es por ello que se debe añadir el archivo **boot_code.c,** para ello seleccionar *Add* y buscar el archivo en la carpeta que se generó previamente con el Nios II SBT (*por lo general se encuentra en la carpeta llamada software y dentro de la carpeta con el nombre de la aplicación*), cabe señalar que el archivo .c no es necesario hacerlo en Nios II SBT, de hecho la utilización del Altera Monitor reemplaza el uso de ese programa, sin embargo debido a que ya se cuenta con ese archivo se lo utiliza para fines demostrativos. Clic *Next*

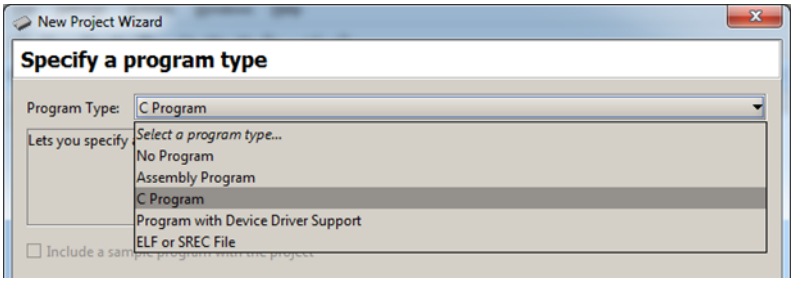


Figura 15. Selección del tipo de archivo que contiene la aplicación de software, creación de proyecto

21. En la siguiente pantalla verificar las configuraciones del sistema como la conexión USB-Blaster y los demás parámetros que se crearon con la herramienta Qsys, como el nombre del procesador, los vectores de reset y excepción. Clic *Next.* En la siguiente y última pantalla verificar las configuraciones de memoria que debn corresponder al componente onchip_memory_SRAM que se añadió en la etapa de desarrollo de hardware. Clic *Finish.*

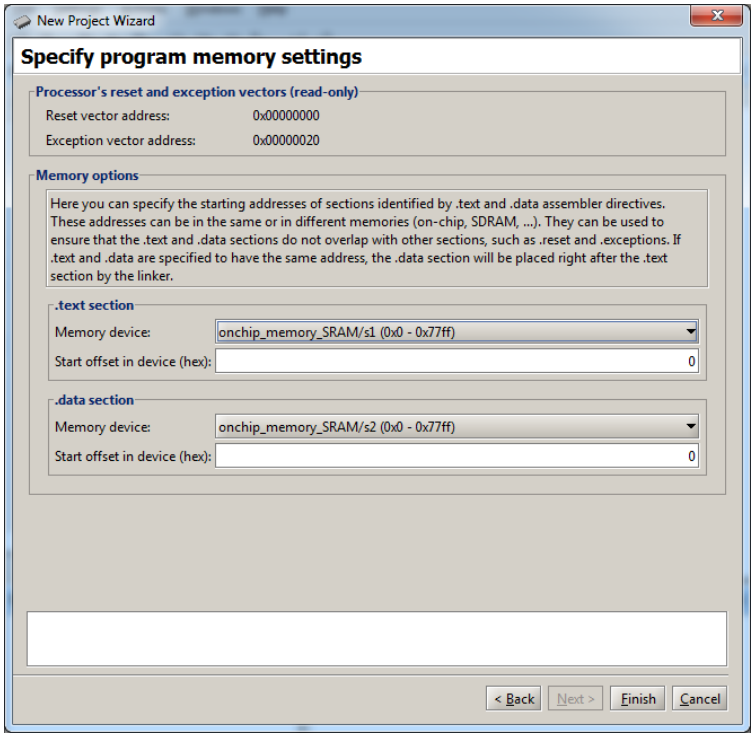


Figura 16. Especificaciones del uso de memoria del programa, creación de proyecto

22. Una vez terminado se confirma que se desea descargar el sistema a la tarjeta.

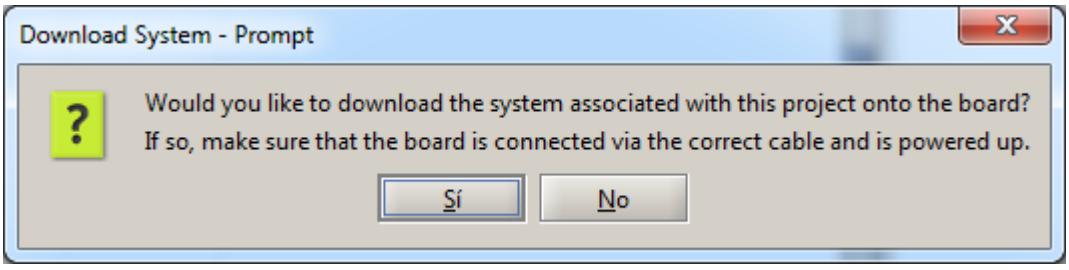


Figura 17. Descargar el sistema a la tarjeta

23. Clic en la pestaña **Actions > Compile & Load**

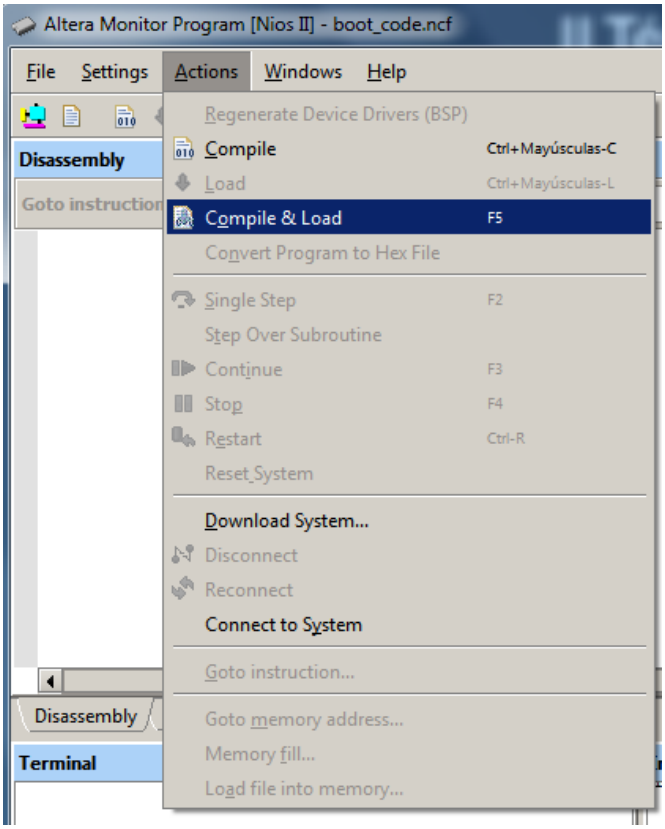


Figura 18. Compilación y descarga del programa

Como resultado de la compilación se crea un archivo ejecutable .elf, similar a cuando se trabaja en Nios II SBT, el Altera Monitor Program descarga este archivo en la on_chip_SRAM.

Para que este archivo ejecutable sea el programa de arranque debe cambiarse a formato .hex y almacenarse en la memoria no volátil EPCS64 de la tarjeta DE0-Nano.

24. En el Altera Monitor Program seleccionar **Actions>Convert Program to Hex File**. Verificar que en el directorio del proyecto se crea el archivo boot_code.hex.

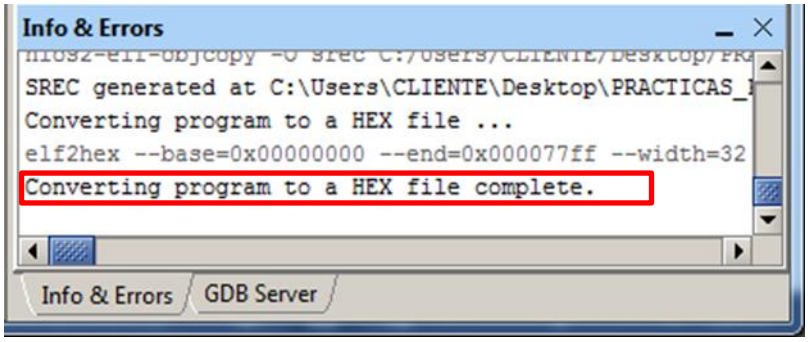


Figura 19. Conversión del tipo de archivo a .hex desde Altera Monitor Program

25. Cerrar el Altera Monitor Program y recompilar el proyecto en Quartus II para crear un nuevo archivo .sof el mismo que contiene la información de que el archivo boot_code.hex deberá ser usado para inicializar la onchip_memory_SRAM (esto se especificó al configurar este componente en Qsys, *véase paso 4 de la presente práctica*).

26. Para programar la memoria flash de la tarjeta DE0-Nano se utiliza un método llamado *JTAG Indirect Configuration,* para ello en Quartus II seleccionar **File>Convert Programming Files**. Seleccionar en el tipo de archivo de programación el .jic y en dispositivo EPCS64 de la opción **"Configuration Device"**.

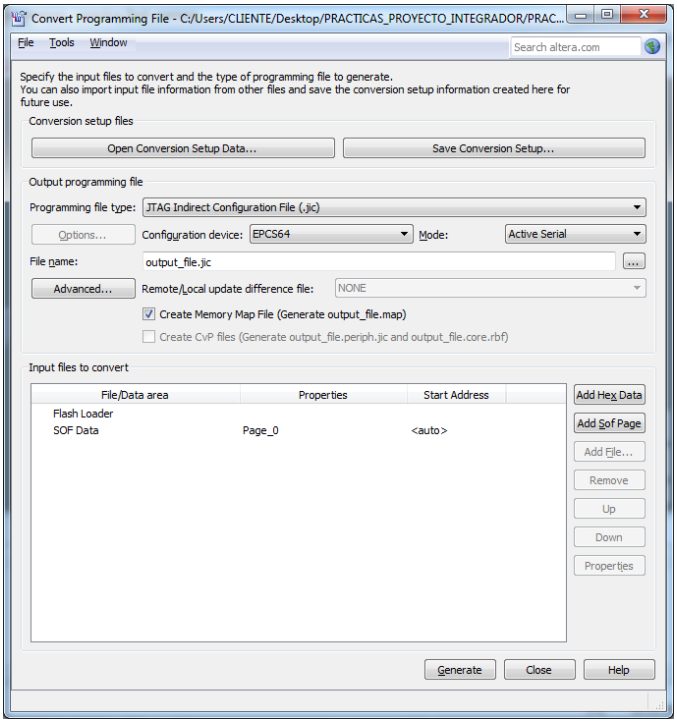


Figura 20. Configuraciones JTAG Indirect Configuration File

Señalar la línea **Flash Loader** y seleccionar **Add Device**. En la ventana que se abre especificar la familia de la FPGA de la De0-Nano, Cyclone IV E y el nombre EP4CE22.

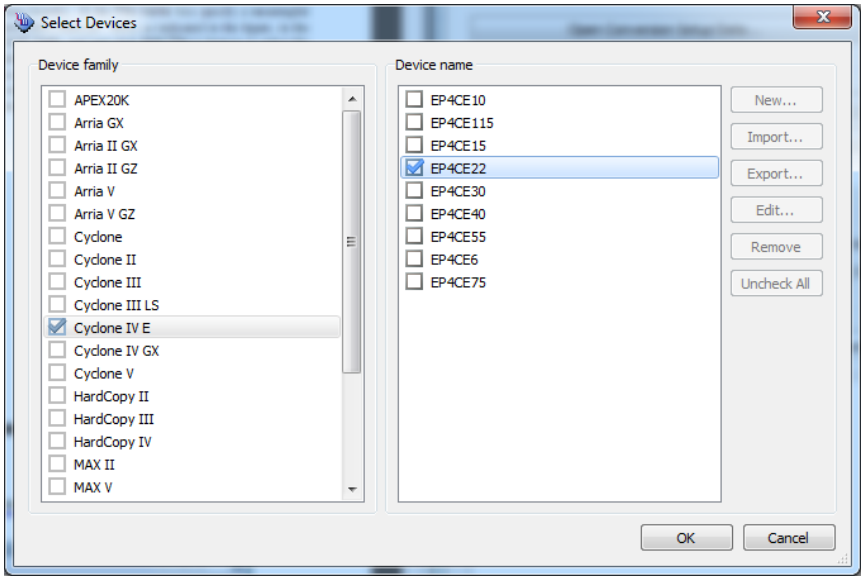


Figura 21. Selección de dispositivo, configuraciones para cargar en la EPCS64

Señalar la línea SOF Data y seleccionar *Add* File, buscar el archivo .sof generado previamente. Señalar el archivo añadido, clic en *Properties* y seleccionar *Compression*. Finalmente clic en *Generate.*

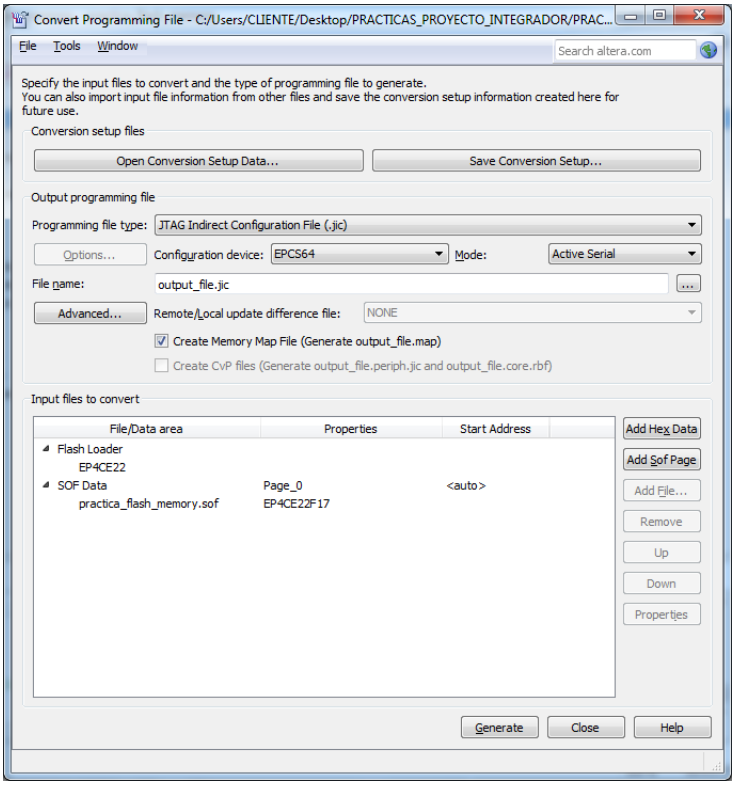


Figura 22. Configuraciones finales para cargar sistema en la EPCS64

27. Ir a la herramienta Programmer, borrar los archivos .sof existentes y añadir el .jic que se encuentra en la carpeta de archivos de salida en el directorio del proyecto, seleccionar las casillas Program/Configure, Verify y Blank-Check. Clic Start.

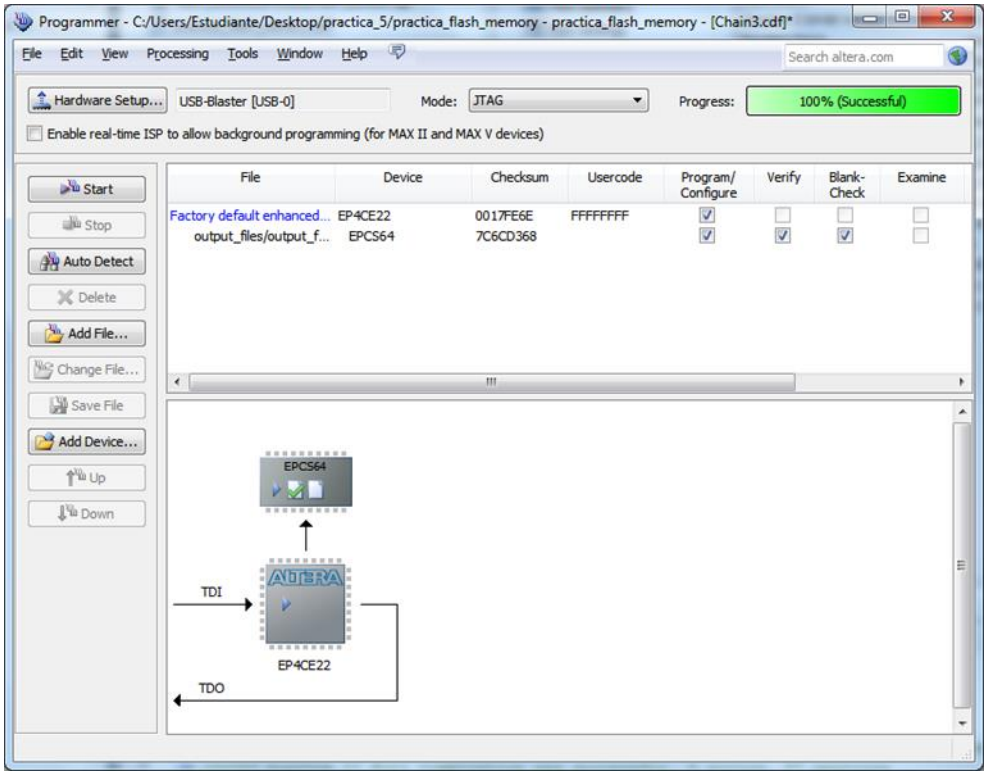


Figura 23. Programado del sistema en el chip integrado en la tarjeta DE0-Nano EPCS64

Una vez descargado el archivo en la memoria flash, al desconectar y conectar la tarjeta se podrá observar que la misma arranca con el sistema que se le cargó, tanto el hardware como la aplicación de software.

PRÁCTICA # 6

TEMA: *Comunicación serial RS-232 entre un sistema embebido sobre la FPGA de la tarjeta DE0-Nano y Proteus.*

1. Objetivos:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Entender y explicar el flujo de datos en una comunicación RS-232, satisfaciendo los requerimientos de voltaje de dicho estándar.
- Proporcionar el hardware requerido en la herramienta Qsys para establecer la comunicación serial desde la tarjeta DE0-Nano.
- Ejecutar una aplicación de software sobre el hardware desarrollado utilizando *'Nios II Software Build for Eclipse',* para el intercambio de caracteres entre la tarjeta DE0-Nano y Proteus.

2. Requisitos mínimos:

- Tener instalado Quartus II y Nios II EDS (versión 13.0).
- Conocimientos básicos de Quartus II y programación en VHDL y C.
- Haber culminado satisfactoriamente la Práctica # 2.
- Lectura y comprensión del archivo: '*FUNDAMENTACIÓN_TEÓRICA_PRÁCTICA_UART_RS232*'

3. Breve explicación de la práctica:

Se desarrolla un sistema de hardware que incluye un procesador embebido de Altera Nios II en conjunto con otros módulos, entre los cuales se encuentra el componente UART (RS-232 Serial Port) *(véase figura 1)*, en el cual se configuran los parámetros de la comunicación serial desde el lado de la tarjeta DE0-Nano. Todos los componentes se conectan por medio de una interconexión llamada *Avalon switch fabric.* De esta manera se forma un sistema que luego se implementa en el chip FPGA utilizando el programa de *Quartus II*.

Todas las partes del sistema basado en el procesador Nios II, son definidas usando lenguaje de descripción de hardware utilizando la herramienta *Qsys* para implementar el sistema deseado simplemente seleccionando los componentes requeridos y especificando los parámetros necesarios. Como memoria del programa y de datos se utiliza el chip SDRAM embebido en la tarjeta DE0-Nano para evitar desbordes de memoria que pueden darse en caso de utilizarse la On-Chip Memory.

Sobre el diseño de hardware desarrollado se corre un programa creado *en Nios II Software Build Tools for Eclipse,* que permite mostrar en consola los caracteres recibidos desde proteus y enviar caracteres desde la tarjeta DE0- Nano, que serán visualizados en un terminal virtual desde el otro extremo de la comunicación serial.

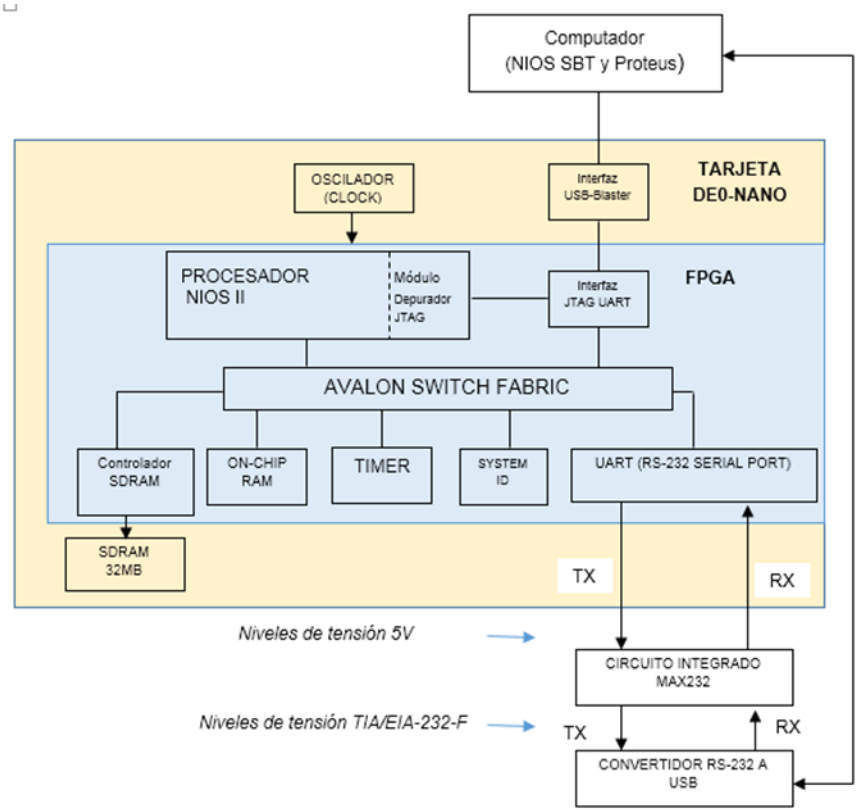


Figura 1. Diagrama de bloques del sistema de hardware a desarrollar en la Práctica # 6

4. Desarrollo de la práctica:

1. Armar el circuito físico requerido y realizar las conexiones para el flujo de datos entre la tarjeta DE0-Nano y el puerto COM físico simulado en Proteus. Se utilizará los pines de propósito general GPIO (pines: 6, 8, 11, 12) del banco JP2 de la tarjeta.

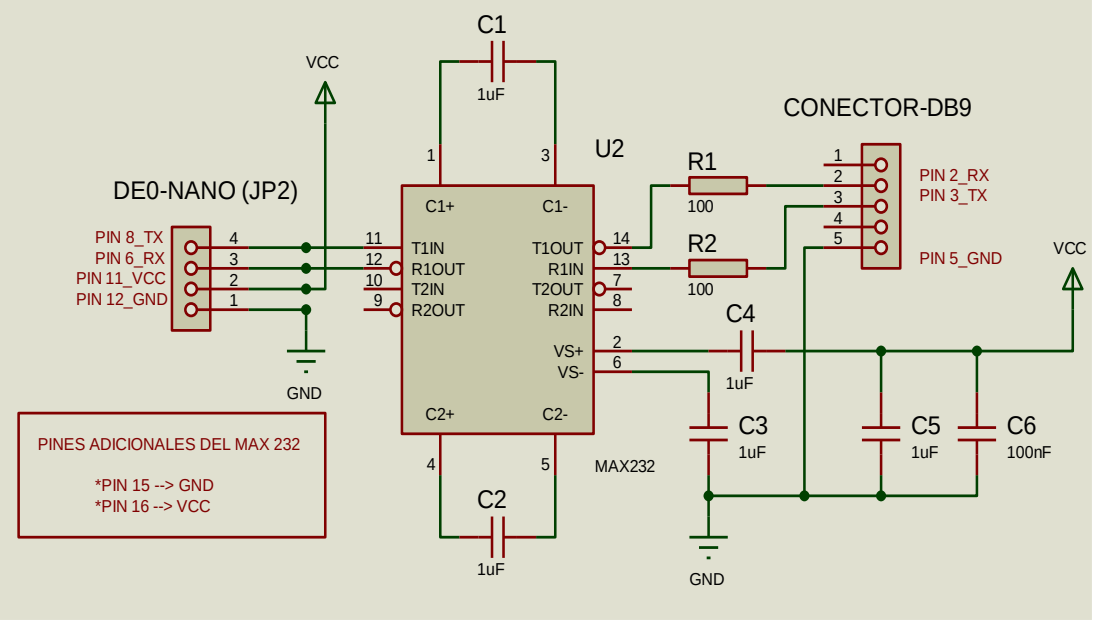


Figura 2. Circuito requerido en la Práctica # 6

2. Abrir el programa Quartus II y crear un proyecto nuevo (*Nombre del proyecto sugerido: practica_serial)* con la familia *Cyclone IV E, dispositivo EP4CE22F17C6.*

3. Seleccionar **Tools > Qsys.**

4. Añadir los componentes básicos del sistema desde la librería, para ello seleccionar y agregar los que se describen a continuación con sus respectivas configuraciones:

Componente	**Renombrar componentes**	**Configuración**
Nios II Processor	cpu	Nios II/s
On-Chip Memory (RAM or ROM)	onchip_mem	default
System ID Peripherial	sysid	default

JTAG UART	jtag_uart	default
Interval Timer	timer	default
SDRAM Controller	sdram	Data Width = 16 bits Chip Select=1, Banks=4 Row= 13, Columns=9
Clock Signals for DE-series Board Peripherals	clocks	DE-Series Board = DE0-Nano Optional Clocks → Sólo señalar SDRAM

Tabla 1. Componentes básicos del sistema de hardware de la práctica # 6

En caso de requerir procedimientos más detallados para agregar los componentes antes mencionados refiérase a la práctica 1 que contiene parte introductoria al uso de Qsys para el desarrollo de sistemas embebidos en el procesador Nios II y a la práctica 2 en el uso de la memoria SDRAM.

5. Añadir el componente UART, para ello en la librería de componentes seguir la ruta **Interface Protocols** > **Serial > UART (RS-232 Serial Port)** y dar doble clic (o en su defecto **Add**).

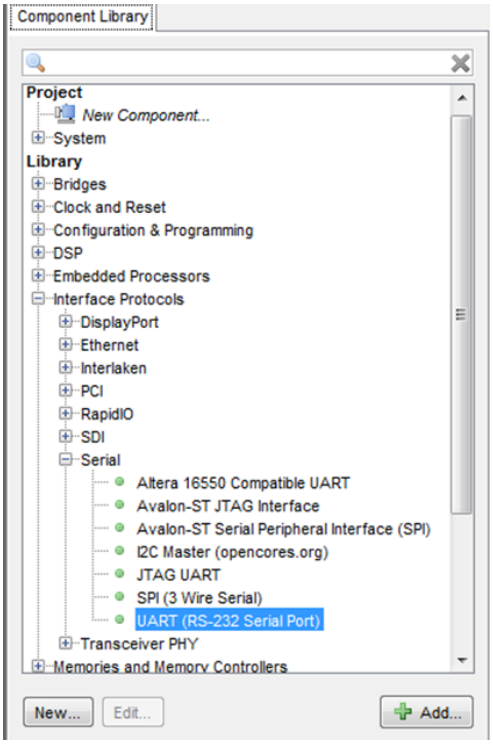


Figura 3. Selección componente UART (RS-232 Serial Port)

Luego de añadirlo se abre automáticamente la ventana de configuraciones del componente, fijamos la velocidad en baudios en 19200 bps y marcamos la casilla fixed baud rate, porque no cambiaremos por código este parámetro.

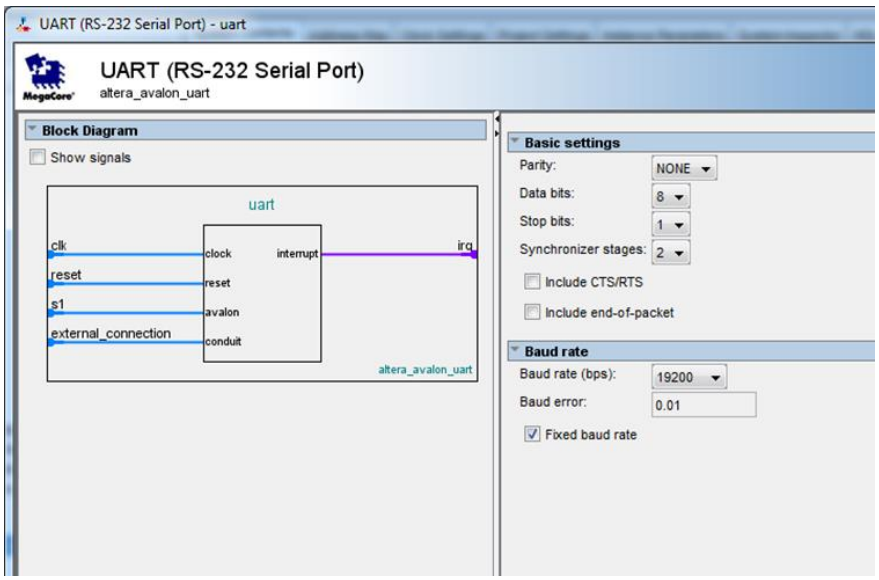


Figura 4. Ajuste de los parámetros del componente UART (RS-232 Serial Port)

6. Recordar que los errores que se visualizan se deben a la falta de conexiones, de direcciones base a los componentes y asignaciones de vectores de memoria de reset y de excepción en el procesador. Las conexiones a realizar se muestran en la *Figura 5*.

7. Se exportan las señales de los componentes a las que posteriormente se le realizarán asignación de pines de la FPGA para ello en la columna **Export** se da doble clic sobre las señales a exportar y se cambia los nombres como se muestra en la *Figura 5*.

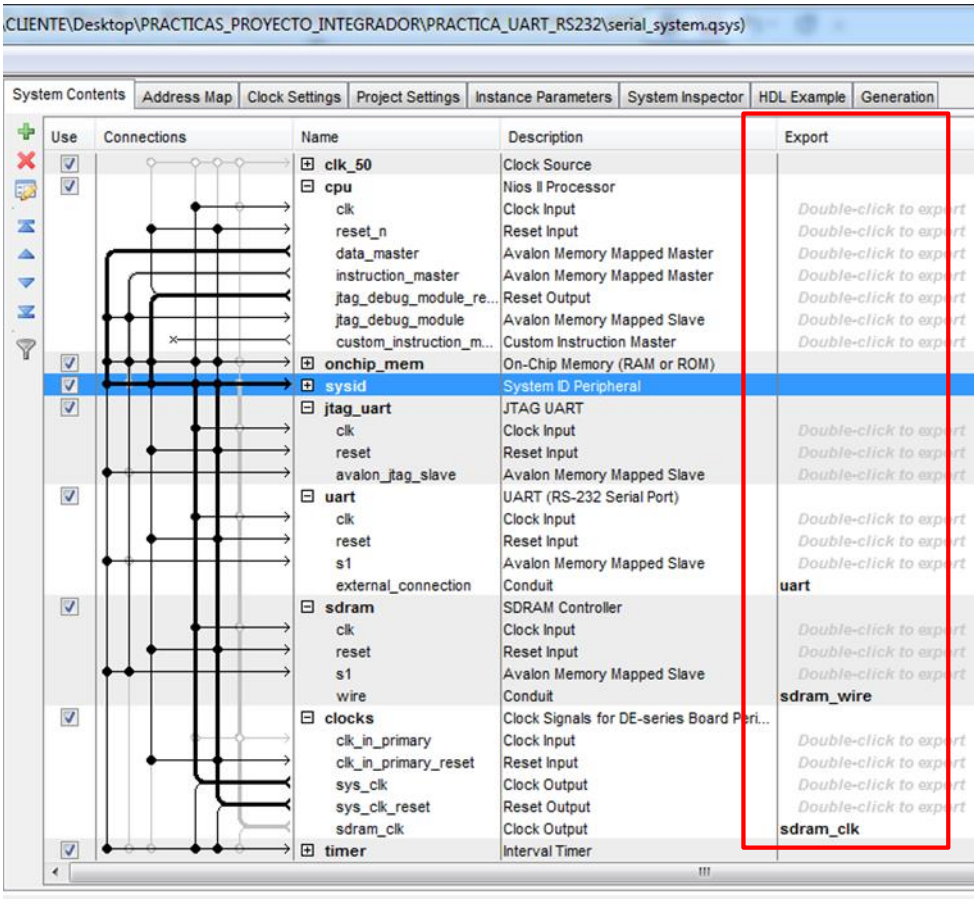


Figura 5. Conexiones entre los componentes del sistema de hardware y señales exportadas

8. En las configuraciones del componente cpu, seleccionar el sdram.s1 como el reset vector y el exception vector.

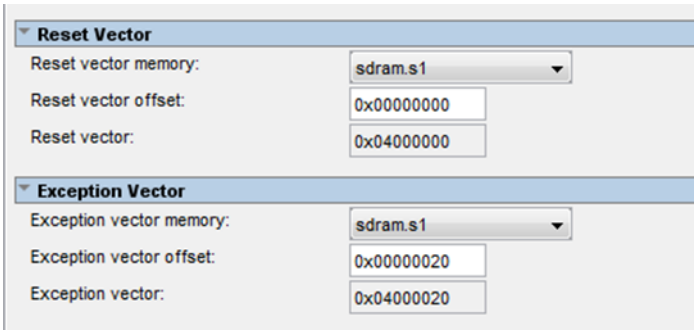


Figura 6. Asignación de los vectores reset y exception

9. Se debe añadir las solicitudes de interrupción o IRQs (Interrupt Request). En la columna IRQ, conectar el Procesador Nios II al JTAG UART, al Interval Timer. Clic en el valor IRQ del jtag_uart y escribir 2, de manera similar escribir 0 para el valor IRQ del componente timer y 1 para el componente UART.

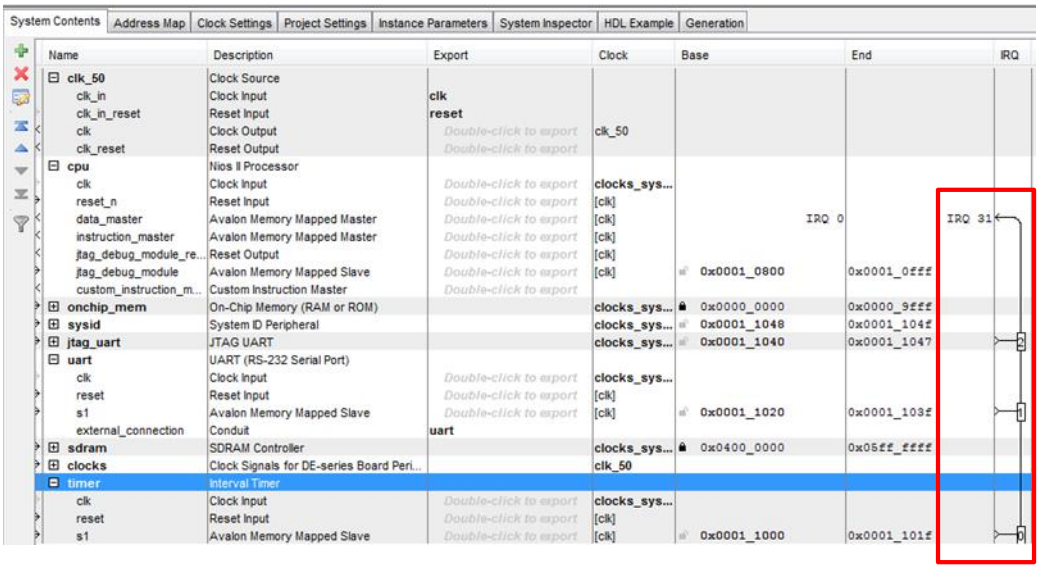


Figura 7. Asignación de la prioridad de interrupciones

10. Añadir las direcciones base, como sugerencia establecer manualmente la dirección base de la on-chip memory a 0x00000000 y fijarlo a ese valor seleccionando el candado ubicado a lado izquierdo de la dirección, de manera

similar para el controlador SDRAM fijar la dirección 0x04000000 y para todos los demás componentes generar las direcciones de forma automática mediante la opción *Assign Base Addresses* en la pestaña System.

11. Guardar, llenar el nombre de archivo (sugerido serial_system). Clic en la pestaña **Generation** y seleccionar None tanto en la sección **Simulation** como en **Testbench System.** Clic **Generate,** al finalizar clic en **Close,** cerrar Qsys y regresar a Quartus II.

12. Se procede a añadir el archivo de extensión .qip generado en Qsys, que contiene la información del sistema Nios II creado. En la ventana de Project Navigator, dar doble clic sobre la carpeta Files, y buscar el archivo que se encuentra en la ruta: <nombre deldirectorio>/serial_system/synthesis/serial_system.qip(Verificar que se esté mostrando todos los archivos o los de extensión .qip al buscar).

13. Se procede a instanciar el módulo del sistema creado en Qsys, para ello crear un nuevo *Block Diagram / Schematic File*, y añadir el bloque serial_system. Agregar los terminales de entrada y salida, con sus respectivas etiquetas. Guardar el archivo (practica_serial).

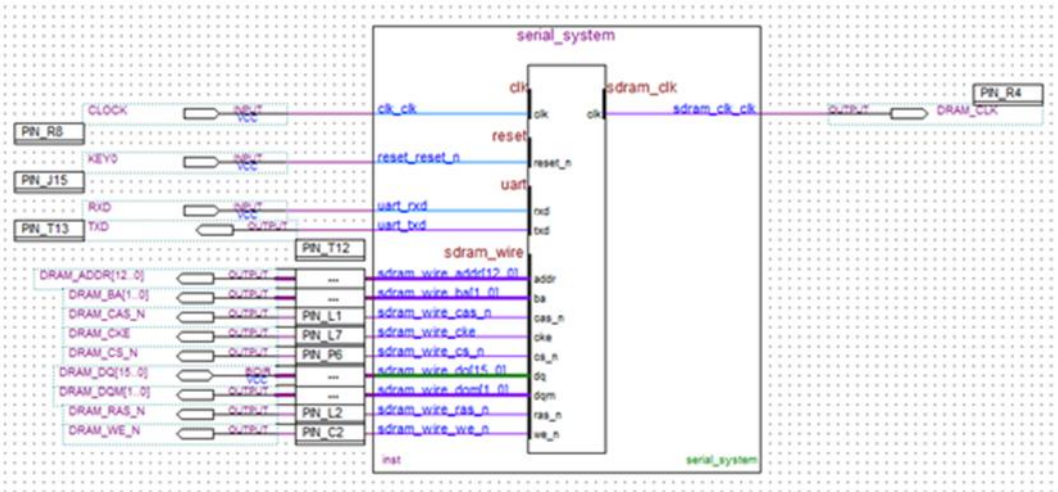


Figura 8. Adición de los pines al nuevo bloque serial_system

14. En el menú **Processing** seleccionar **Start** > **Start Analysis & Elaboration,** con lo cual al revisar el **Pin Planner** observamos listadas las entradas y salidas creadas.

15. Se realizan las asignaciones de pines utilizada en la práctica # 2 del uso de la SDRAM y para los pines RXD y TXD se le asignan los GPI/O T13 y T12 respectivamente. En el directorio del proyecto existe un archivo con la extensión .qsf, el mismo que contiene la asignación de pines, en caso de tener en digital la asignación de pines puede añadirse a la parte ya existente. Se puede copiar el siguiente código en el archivo .qsf ubicado en el directorio similar a las prácticas previas. Para una operación apropiada debe cambiarse el estándar, en donde dice “2.5 V” cambiar por “3.3-V LVTTL”. Guarde los cambios del archivo y ciérrelo.

```
set_global_assignment -name STRATIX_DEVICE_IO_STANDARD "3.3-V LVTTL"
set_location_assignment PIN_R8 -to CLOCK
set_location_assignment PIN_J15 -to KEY0
set_location_assignment PIN_T13 -to RXD
set_location_assignment PIN_T12 -to TXD
set_location_assignment PIN_L4 -to DRAM_ADDR[12]
set_location_assignment PIN_N1 -to DRAM_ADDR[11]
set_location_assignment PIN_N2 -to DRAM_ADDR[10]
set_location_assignment PIN_P1 -to DRAM_ADDR[9]
set_location_assignment PIN_R1 -to DRAM_ADDR[8]
set_location_assignment PIN_T6 -to DRAM_ADDR[7]
set_location_assignment PIN_N8 -to DRAM_ADDR[6]
set_location_assignment PIN_T7 -to DRAM_ADDR[5]
set_location_assignment PIN_P8 -to DRAM_ADDR[4]
set_location_assignment PIN_M8 -to DRAM_ADDR[3]
set_location_assignment PIN_N6 -to DRAM_ADDR[2]
set_location_assignment PIN_N5 -to DRAM_ADDR[1]
set_location_assignment PIN_P2 -to DRAM_ADDR[0]
set_location_assignment PIN_M6 -to DRAM_BA[1]
set_location_assignment PIN_M7 -to DRAM_BA[0]
set_location_assignment PIN_L1 -to DRAM_CAS_N
set_location_assignment PIN_L7 -to DRAM_CKE
```

```
set_location_assignment PIN_R4 -to DRAM_CLK
set_location_assignment PIN_P6 -to DRAM_CS_N
set_location_assignment PIN_K1 -to DRAM_DQ[15]
set_location_assignment PIN_N3 -to DRAM_DQ[14]
set_location_assignment PIN_P3 -to DRAM_DQ[13]
set_location_assignment PIN_R5 -to DRAM_DQ[12]
set_location_assignment PIN_R3 -to DRAM_DQ[11]
set_location_assignment PIN_T3 -to DRAM_DQ[10]
set_location_assignment PIN_T2 -to DRAM_DQ[9]
set_location_assignment PIN_T4 -to DRAM_DQ[8]
set_location_assignment PIN_R7 -to DRAM_DQ[7]
set_location_assignment PIN_J1 -to DRAM_DQ[6]
set_location_assignment PIN_J2 -to DRAM_DQ[5]
set_location_assignment PIN_K2 -to DRAM_DQ[4]
set_location_assignment PIN_K5 -to DRAM_DQ[3]
set_location_assignment PIN_L8 -to DRAM_DQ[2]
set_location_assignment PIN_G1 -to DRAM_DQ[1]
set_location_assignment PIN_G2 -to DRAM_DQ[0]
set_location_assignment PIN_L2 -to DRAM_RAS_N
set_location_assignment PIN_C2 -to DRAM_WE_N
set_location_assignment PIN_R6 -to DRAM_DQM[0]
set_location_assignment PIN_T5 -to DRAM_DQM[1]
```

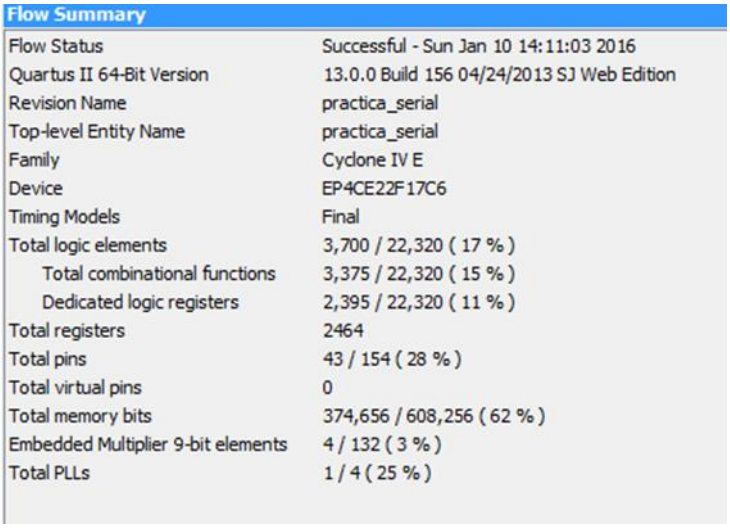
Named: * Edit:

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved
CLOCK	Input	PIN_R8	3	B3_N0	PIN_R8	3.3-V LV...default)	
DRAM_ADDR[12]	Output	PIN_L4	2	B2_N0	PIN_L4	3.3-V LV...default)	
DRAM_ADDR[11]	Output	PIN_N1	2	B2_N0	PIN_N1	3.3-V LV...default)	
DRAM_ADDR[10]	Output	PIN_N2	2	B2_N0	PIN_N2	3.3-V LV...default)	
DRAM_ADDR[9]	Output	PIN_P1	2	B2_N0	PIN_P1	3.3-V LV...default)	
DRAM_ADDR[8]	Output	PIN_R1	2	B2_N0	PIN_R1	3.3-V LV...default)	
DRAM_ADDR[7]	Output	PIN_T6	3	B3_N0	PIN_T6	3.3-V LV...default)	
DRAM_ADDR[6]	Output	PIN_N8	3	B3_N0	PIN_N8	3.3-V LV...default)	
DRAM_ADDR[5]	Output	PIN_T7	3	B3_N0	PIN_T7	3.3-V LV...default)	
DRAM_ADDR[4]	Output	PIN_P8	3	B3_N0	PIN_P8	3.3-V LV...default)	
DRAM_ADDR[3]	Output	PIN_M8	3	B3_N0	PIN_M8	3.3-V LV...default)	
DRAM_ADDR[2]	Output	PIN_N6	3	B3_N0	PIN_N6	3.3-V LV...default)	
DRAM_ADDR[1]	Output	PIN_N5	3	B3_N0	PIN_N5	3.3-V LV...default)	
DRAM_ADDR[0]	Output	PIN_P2	2	B2_N0	PIN_P2	3.3-V LV...default)	
DRAM_BA[1]	Output	PIN_M6	3	B3_N0	PIN_M6	3.3-V LV...default)	
DRAM_BA[0]	Output	PIN_M7	3	B3_N0	PIN_M7	3.3-V LV...default)	
DRAM_CAS_N	Output	PIN_L1	2	B2_N0	PIN_L1	3.3-V LV...default)	
DRAM_CKE	Output	PIN_L7	3	B3_N0	PIN_L7	3.3-V LV...default)	
DRAM_CLK	Output	PIN_R4	3	B3_N0	PIN_R4	3.3-V LV...default)	
DRAM_CS_N	Output	PIN_P6	3	B3_N0	PIN_P6	3.3-V LV...default)	
DRAM_DQ[15]	Bidir	PIN_K1	2	B2_N0	PIN_K1	3.3-V LV...default)	
DRAM_DQ[14]	Bidir	PIN_N3	3	B3_N0	PIN_N3	3.3-V LV...default)	
DRAM_DQ[13]	Bidir	PIN_P3	3	B3_N0	PIN_P3	3.3-V LV...default)	
DRAM_DQ[12]	Bidir	PIN_R5	3	B3_N0	PIN_R5	3.3-V LV...default)	
DRAM_DQ[11]	Bidir	PIN_R3	3	B3_N0	PIN_R3	3.3-V LV...default)	
DRAM_DQ[10]	Bidir	PIN_T3	3	B3_N0	PIN_T3	3.3-V LV...default)	
DRAM_DQ[9]	Bidir	PIN_T2	3	B3_N0	PIN_T2	3.3-V LV...default)	
DRAM_DQ[8]	Bidir	PIN_T4	3	B3_N0	PIN_T4	3.3-V LV...default)	
DRAM_DQ[7]	Bidir	PIN_R7	3	B3_N0	PIN_R7	3.3-V LV...default)	
DRAM_DQ[6]	Bidir	PIN_J1	2	B2_N0	PIN_J1	3.3-V LV...default)	
DRAM_DQ[5]	Bidir	PIN_J2	2	B2_N0	PIN_J2	3.3-V LV...default)	
DRAM_DQ[4]	Bidir	PIN_K2	2	B2_N0	PIN_K2	3.3-V LV...default)	
DRAM_DQ[3]	Bidir	PIN_K5	2	B2_N0	PIN_K5	3.3-V LV...default)	
DRAM_DQ[2]	Bidir	PIN_L8	3	B3_N0	PIN_L8	3.3-V LV...default)	
DRAM_DQ[1]	Bidir	PIN_G1	1	B1_N0	PIN_G1	3.3-V LV...default)	
DRAM_DQ[0]	Bidir	PIN_G2	1	B1_N0	PIN_G2	3.3-V LV...default)	
DRAM_DQM[1]	Output	PIN_T5	3	B3_N0	PIN_T5	3.3-V LV...default)	
DRAM_DQM[0]	Output	PIN_R6	3	B3_N0	PIN_R6	3.3-V LV...default)	
DRAM_RAS_N	Output	PIN_L2	2	B2_N0	PIN_L2	3.3-V LV...default)	
DRAM_WE_N	Output	PIN_C2	1	B1_N0	PIN_C2	3.3-V LV...default)	
KEY0	Input	PIN_J15	5	B5_N0	PIN_J15	3.3-V LV...default)	
RXD	Input	PIN_T13	4	B4_N0	PIN_T13	3.3-V LV...default)	
TXD	Output	PIN_T12	4	B4_N0	PIN_T12	3.3-V LV...default)	
<<new node>>							

All Pins

Figura 7. Pines asignados a cada señal exportada

16. Compilar el proyecto para crear el archivo .sof que se descargará en la tarjeta, luego de lo cual podemos observar el reporte de compilación que fue exitoso y la utilización de elementos lógicos de la FPGA, también muestra el uso de la memoria contenida en la FPGA (On-chip Memory), que está al 62%, y la misma en esta práctica únicamente maneja datos temporales y provee un control del flujo del sistema; pero no se la utiliza como memoria del programa ni de datos del programa a compilar sobre el hardware diseñado.



Flow Summary	
Flow Status	Successful - Sun Jan 10 14:11:03 2016
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	practica_serial
Top-level Entity Name	practica_serial
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	3,700 / 22,320 (17 %)
Total combinational functions	3,375 / 22,320 (15 %)
Dedicated logic registers	2,395 / 22,320 (11 %)
Total registers	2464
Total pins	43 / 154 (28 %)
Total virtual pins	0
Total memory bits	374,656 / 608,256 (62 %)
Embedded Multiplier 9-bit elements	4 / 132 (3 %)
Total PLLs	1 / 4 (25 %)

Figura 8. Reporte de compilación del sistema de hardware

17. Descargar el archivo .sof en la tarjeta DE0-Nano con la herramienta de programación (**Tools > Programmer**).

18. Abrir el programa Nios II SBT desde quartus en la pestaña de herramientas (**Tools > Nios II Software Build Tools for Eclipse**) y se procede a la creación del software a correr sobre el hardware diseñado previamente, para ello crear una carpeta en el directorio del proyecto con el nombre workspace (sugerido). Crear un proyecto en blanco (nombre: uart_rs232), seleccionando el correspondiente archivo .sopcinfo. Al proyecto creado añadir un archivo main.c, en el mismo se escribirá el siguiente código:

```
#include <stdio.h>
#include <string.h>

int main ()
{
	char* msg1 = "O";
	char* msg2 = "K";
	char* msg3 = " ";
FILE* fp;
char caracter;
int count=0;
	while (1)
```

```
	{
		fp = fopen ("/dev/uart", "r+"); //Open file for reading
and writing
		count=count++;
		caracter= getc(fp); // Get a character from the UART.
		printf("Letra %d = %c \n",count, caracter);
		fwrite (msg1, strlen (msg1), 1, fp);
		fclose (fp);
		usleep(10000);
		fp = fopen ("/dev/uart", "r+"); //Open file for reading
and writing
		fwrite (msg2, strlen (msg2), 1, fp);
		fclose (fp);
		usleep(10000);
		fp = fopen ("/dev/uart", "r+"); //Open file for reading
and writing
		fwrite (msg3, strlen (msg3), 1, fp);
		fclose (fp);
	}

return 0;

}
```

19. Clic derecho sobre la carpeta del proyecto de aplicación (no sobre el bsp) y seleccionar **Build Project**, nuevamente clic derecho sobre el proyecto y seleccionar **Run As > Nios II Hardware**. Una vez que termina de cargar el programa, abrir un proyecto esquemático en proteus, añadir un conector DB9 (buscarlo en componentes con el nombre COMPIM) y dos terminales virtuales e interconectarlos como se muestra a continuación.

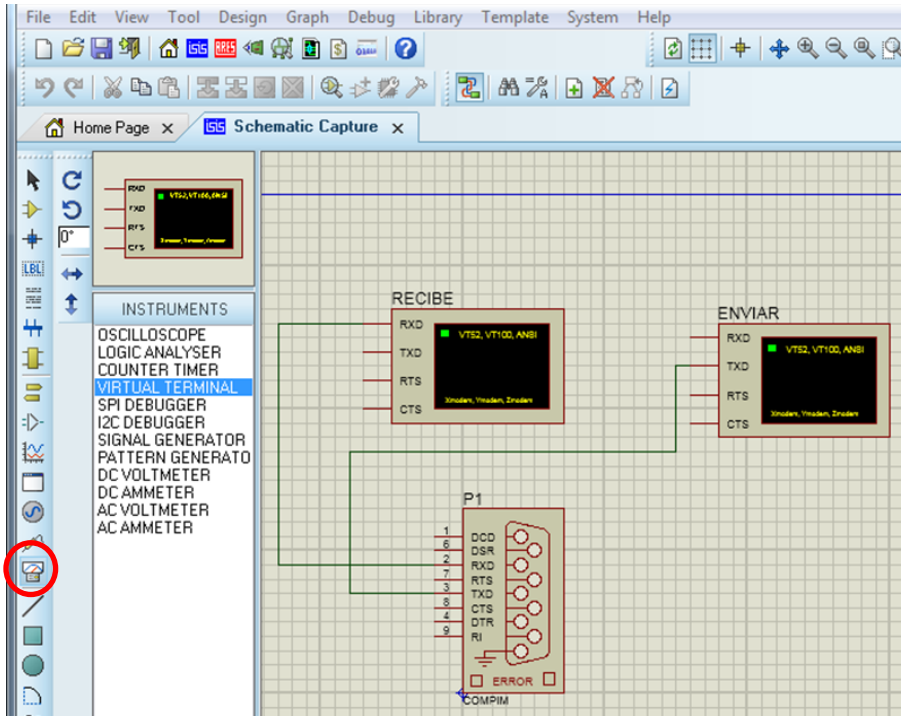


Figura 9. Adición de conector DB9 y terminales virtuales en proteus

20. Dar doble clic sobre el componente COMPIM y realizar la configuración de los parámetros que deben coincidir con los que se fijaron al crear el componente UART en Qsys.

    Para fijar el valor del parámetro *Physical port, se* debe conectar el cable convertidor RS-232 a USB a la computadora y verificar en que puerto se encuentra en el administrador de dispositivos de la máquina.

Figura 10. Localización del puerto asignado al convertidos USB-RS232 por la computadora

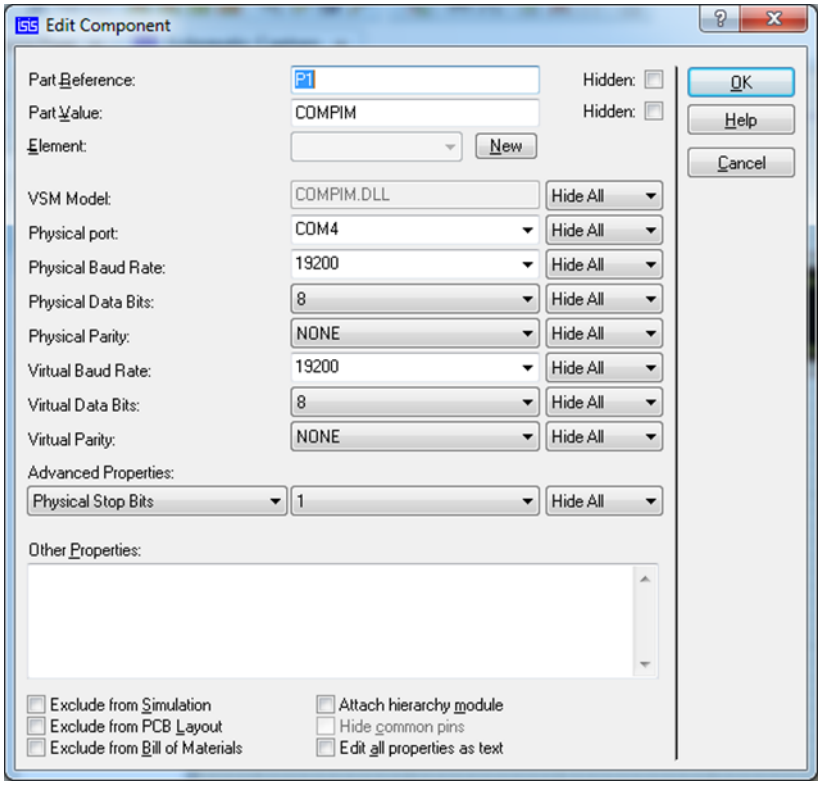


Figura 11. Ajuste de los parámetros del componente COMPIM en proteus

21. Guardar y correr el archivo de Proteus, con ello deben abrirse las dos ventanas de los componentes virtual terminal, una para enviar y otra para recibir datos.

Al escribir un carácter en el terminal de **ENVIAR** desde proteus, en la consola de Nios SBT se muestra el carácter recibido y se envía “OK ” que se visualiza en el terminar de RECIBE en Proteus, así verificamos que la comunicación se ha establecido correctamente.

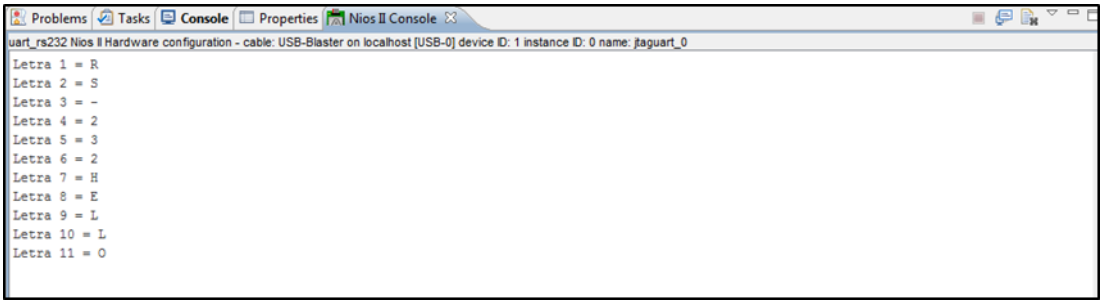


Figura 12. Consola de Nios II SBT

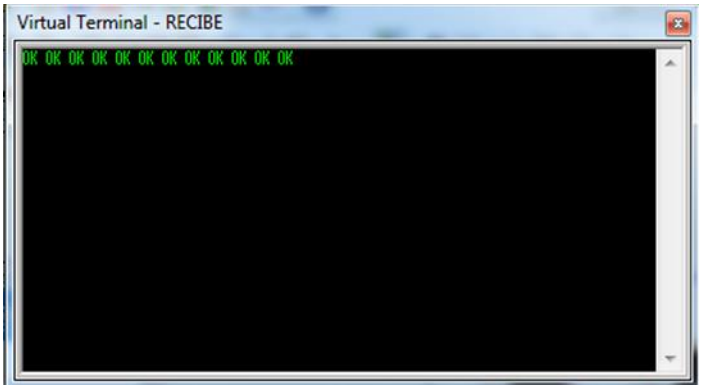


Figura 13. Terminal virtual de Proteus al recibir datos

Figura 14. Terminal virtual de Proteus para enviar datos, los mismos no quedan registrados

El código que se corrió sobre el sistema Nios II, hace uso de librerías C estándares para la lectura y escritura de caracteres al componente *uart*. A continuación se procede a establecer la comunicación serial fijando el componente uart como salida y entrada estándar (*stdout y stdin*) que por defecto están asignados al componente jtag-uart y por lo cual al utilizar funciones como printf(), se muestra en la consola de Nios II SBT como en el código utilizado anteriormente en la práctica, que se imprimía "Letra # = _" en dicha consola.

22. Crear un nuevo proyecto (nombre sugerido rs232_app) en Nios II SBT (no es necesario borrar o cerrar el anterior) y escribir la siguiente aplicación en un archivo de código fuente (*por ejemplo en main.c*):

```
#include <stdio.h>

int main()
{
	int recibido;

	while (1){
	scanf("%c",recibido);
	printf("O");
	usleep(10000);
	printf("K");
	usleep(10000);
	printf(" ");
	usleep(1000000);
	}
  return 0;

}
```

23. En el proyecto bsp creado recientemente dar clic derecho y seleccionar **Nios II > BSP Editor**. Configurar la librería hal del sistema, para ello en la ventana que se abrió, en la pestaña Main > sección hal, cambiar los valores de stderr, stdin y stdout del valor por defecto a uart. Clic **Generate.**

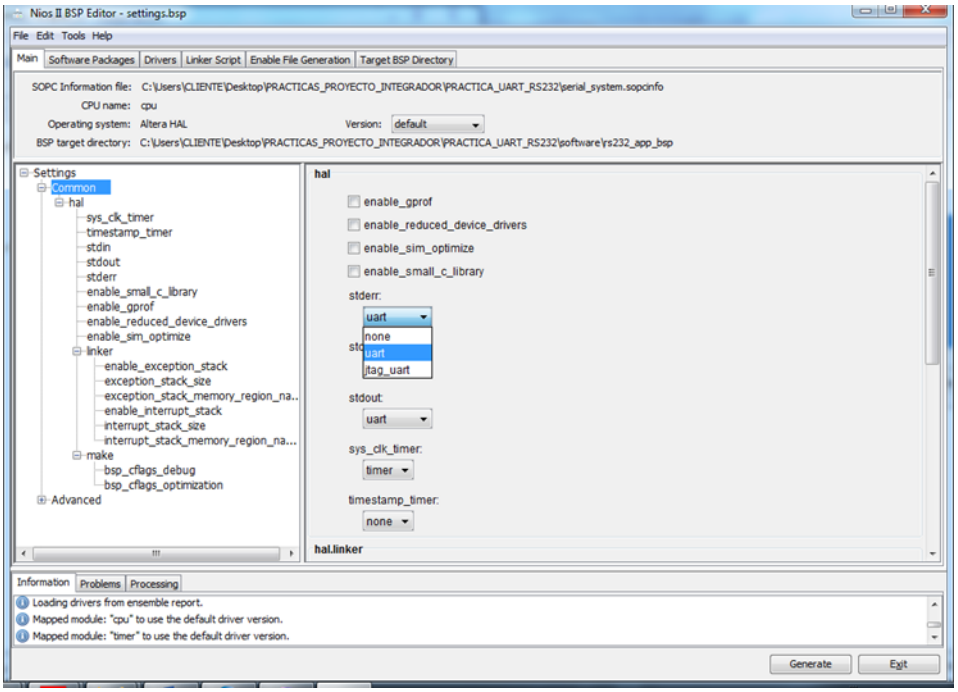


Figura 15. Configuración de librería hal del sistema

24. Clic derecho sobre la carpeta del proyecto de aplicación y seleccionar **Build Project**, nuevamente clic derecho sobre el proyecto y seleccionar **Run As > Nios II Hardware**. Una vez que termina de cargar el programa, abrir el proyecto esquemático de Proteus y seleccionar Run (en caso de que aún haya estado corriendo detenerlo y volver a dar Run).

    Cada vez que se presione una tecla en el vitual terminal ENVIAR el procesador la recibe y envía la respuesta OK al virtual terminal RECIBE, haciendo uso de las funciones scanf () y printf ().

Figura 16. Verificación de recepción de datos en la terminal virtual de Proteus

Anexo 2: Fundamentación teórica de cada práctica

FUNDAMENTACIÓN TEÓRICA PRÁCTICA # 1

TEMA: Introducción al diseño de hardware y creación de software para un sistema basado en el procesador Nios II utilizando la tarjeta de desarrollo DE0-Nano.

Tarjeta de desarrollo DE0-Nano

La tarjeta DE0-Nano introduce una plataforma de desarrollo de FPGA de tamaño compacto adecuado para una amplia gama de diseño de proyectos

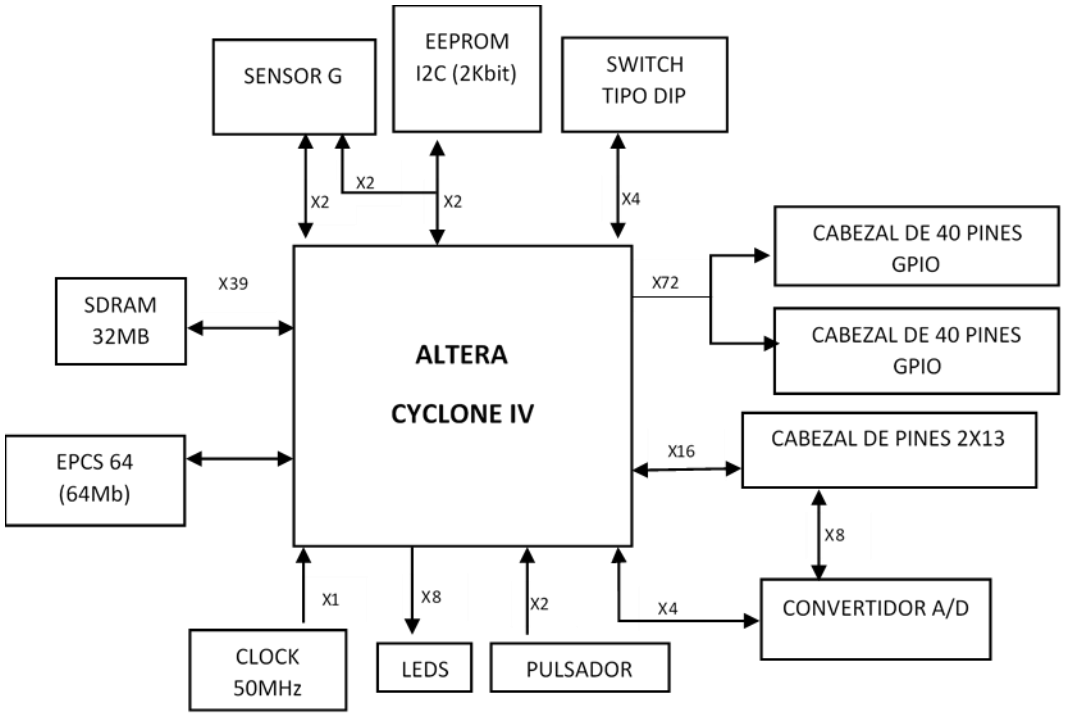


Diagrama de bloques tarjeta DE0-Nano

Herramientas a utilizar de Quartus II

→ Qsys

→ Nios II Software Build Tools (SBT) for Eclipse

Qsys

Es una herramienta que utilizamos para diseñar el sistema de hardware digital, en el mismo se seleccionan los componentes que se desean en el sistema, tales como procesadores, memorias, interfaces de entrada y salida, temporizadores, etc., Al final del diseño se genera el hardware que conecta todos los componentes..

Componentes a utilizar propios de Qsys

• Nios II Core.- Es un módulo de un 'soft-processor'; es decir un procesador que puede ser enteramente implementado utilizando un proceso que convierte un circuito que se tiene en forma abstracta en una implementación de un diseño en términos de puertas lógicas.

Se cuenta con tres alternativas de selección al agregar el procesador:

- Nios II/e (economy).- Está diseñado para conseguir el menor tamaño de núcleo posible. Como resultado este núcleo tiene limitadas características que establecer y muchos ajustes no están disponibles cuando éste es seleccionado.
- Nios II/s (standard).- Este núcleo está diseñado para ser de pequeño tamaño manteniendo el rendimiento.
- Nios II/f (fast).- Está diseñado para un rendimiento rápido. Como resultado este núcleo presenta todas las opciones de configuración permitiendo definir los ajustes para el desarrollo del procesador.

- On-Chip Memory.- Almacena datos de manera temporal y provee un control del flujo en un sistema Qsys. Sistemas de procesadores requieren al menos una memoria para los datos y las instrucciones. La DE0-Nano tiene 594 Kilobits (Kb) de esta memoria embebida en la FPGA, equivalente a 74.25KiloBytes (KB).

- JTAG UART.- Implementa un método para comunicar de forma serial el flujo de caracteres entre la PC y un sistema de Qsys sobre una FPGA de Altera.

- Interval Timer.- La mayoría de los sistemas de control requieren de un temporizador para precisar el cálculo del tiempo. Este bloque provee de contadores de 32 y 64 bits, control de inicio, parada y reset del temporizador, característica opcional de perro guardián que resetea el sistema si el temporizador alcanza cero.

- System ID Peripheral.- Este bloque protege contra accidentalmente descargar software compilado para un sistema de Nios II diferente, para llevar a cabo esta tarea le provee al sistema Qsys con un identificador único.

- PIO (Parallel Input/Output).- Provee un método sencillo para que el procesador del Sistema pueda recibir señales de entradas y manejar las salidas.

Tipos de archivos generados en Qsys más importantes

Se describen los archivos o directorios que se muestran a continuación son aquellos a los cuales se hacen mención en etapas posteriores para el entendimiento o uso en el desarrollo del sistema embebido basado en el procesador Nios II en su totalidad.

Nombre del archivo o del directorio	Descripción
<diseño_qsys>	Es el directorio que contiene todo el proyecto.
<diseño_qsys>**.bsf**	Es la representación del sistema de Qsys en forma de bloque para utilizarlo en Quartus II Block Diagram Files (.bdf)
<diseño_qsys>**.html**	Es un reporte del sistema que contiene información sobre las conexiones externas del sistema, las direcciones de cada componente respecto a las direcciones del maestro y los parámetros asignados para cada componente.
<diseño_qsys>**.sopcinfo**	Describe los componentes y conexiones en el sistema. Este archivo es una completa descripción del sistema.
/synthesis	Este directorio contiene todos los archivos de salida generados por Qsys usados al sintetizar el diseño.
<diseño_qsys>**.v**	Contiene un archivo en HDL del más alto nivel del sistema que instancia cada componente en el sistema.
<diseño_qsys>**.qip**	Este archivo contiene el software de quartus II necesitado para compilar el diseño. Este archivo debe ser añadido al proyecto.
/submodules	Contiene los códigos verilog HDL o VHDL de cada submódulo para la síntesis.

Nios II Software Build Tools (SBT) for Eclipse

Es un conjunto de módulos basados en la plataforma de Eclipse y los módulos de las herramientas de desarrollo de Eclipse C/C++ (CDT). El Nios II SBT provee una plataforma consistente que trabaja para todos los sistemas de procesador embebidos Nios II. Se puede llevar a cabo en este entorno todas las tareas de desarrollo de software tales como creación, edición, construcción, ejecución, depuración y mediciones dinámicas de programas.

Es importante entender que Eclipse es un programa compuesto de un conjunto de herramientas de programación de código abierto que suele ser usado para desarrollar entornos de desarrollo integrado, y justamente esta es la utilidad que le dio Altera para tener su propio entorno de desarrollo de software para sus procesadores Nios II.

Tipos de proyectos de software Nios II a utilizar

Cada programa Nios II que se desarrolla consiste de un proyecto de aplicación, un proyecto de librería de usuario y un proyecto BSP. Al construir el programa Nios II se crea un archivo ejecutable de extensión .elf (Executable and Linking Format File) el mismo que corre sobre el procesador Nios II.

- **Proyecto de Aplicación**

Este proyecto consiste en una colección de código fuente y un makefile, que es el componente principal del proyecto de software Nios II, este archivo describe todos los componentes de software del proyecto y como son compilados y enlazados.

- **Proyecto BSP**

Es una librería especializada que contiene código de soporte específico del sistema. Un BSP contiene los siguientes elementos: librería de funciones primitivas conocidas

como HAL (Hardware Abstraction Layer), librerías estándares de lenguaje C, controladores de los dispositivos, sistema operativo en tiempo real (opcional).

Estructura de interconexión del sistema

La estructura de interconexión del sistema (*System Interconnect Fabric*) es la colección de interconexiones y recursos lógicos que conectan las interfaces de los componentes en un sistema. En la herramienta Qsys que se describió anteriormente se genera la estructura de interconexión del sistema para satisfacer las necesidades de los componentes del mismo. Esta interconexión garantiza que las señales están conectadas correctamente entre el maestro y los esclavos

Fuentes de información:

1. DE0-Nano User Manual v1.9
2. Nios II Classic Software Developer's Handbook
3. Nios II Hardware Development

FUNDAMENTACIÓN TEÓRICA PRÁCTICA # 2

TEMA: *Lectura y escritura sobre el chip SDRAM integrado en la tarjeta DE0-Nano y control de una pantalla LCD 16x2.*

La SDRAM es una memoria volátil, que debe ser refrescada periódicamente para mantener su contenido, además de este requerimiento típicamente necesita el uso de un controlador de hardware especial. La SDRAM divide su memoria en bancos, filas y columnas. Cambiar entre bancos y filas incurre en ciertas sobrecargas, por lo tanto el uso eficiente de esta memoria incluye las cuidadosas órdenes de acceso. Este tipo de memoria también multiplexa las direcciones de filas y columnas sobre las misma línea de dirección.

Las SDRAM son las menos costosas y de mayor capacidad tipos de memoria RAM disponibles, haciéndolos por ello los más populares. La mayoría de los modernos sistemas embebidos usan SDRAM.

Los controladores SDRAM, como el provisto por Qsys, manejan toda la multiplexación de direcciones, refrescado y cambio de filas y bancos, permitiendo al resto del sistema acceder a la memoria sin conocer de su arquitectura interna.

Chip SDRAM de la tarjeta DE0-Nano: IS42S16160B

Es una memoria DRAM (Dynamic Random Access Memory) sincrónica, lo que quiere decir que a diferencia de las clásicas DRAMs espera por una señal de reloj antes de responder a señales de entrada de control. Puede almacenar 256Mb (32MB organizados: 4M x 16 x 4 bancos) y alcanza altas velocidades de transferencia de

datos. Cada uno de los 4 bancos están organizados en 8192 filas (2^13), 512 columnas (2^9) por 16 bits. Todas las señales de entrada y salida están referidas a los flancos de subida del reloj de entrada. En lo que respecta a la alimentación de voltaje utiliza estándar 3.3V LVTTL.

No incurrir en desperdicio de memoria

El maestro de instrucciones del procesador Nios II no puede direccionar más de 256MB de memoria, por lo tanto proveerle más de ellos para correr el software de Nios II es un desperdicio de memoria. En nuestro caso la SDRAM no cae en este plano si la utilizamos para correr en ella el programa (como en la práctica 2), porque tiene 32MB como se indicó anteriormente. Esta restricción no aplica para el maestro de datos de Nios II que puede direccionar tanto como 2GB.

Coincidir el ancho de dato del maestro con el del esclavo

Si el puerto de maestro está conectado con un esclavo que tiene menor ancho de dato, Qsys inserta lógica para traducir entre ellos, esta lógica adicional puede provocar mayor latencia entre cada transacción. Lo mejor es tratar de mantener el ancho de dato del maestro y esclavo iguales para que no impacte negativamente en el desempeño del sistema.

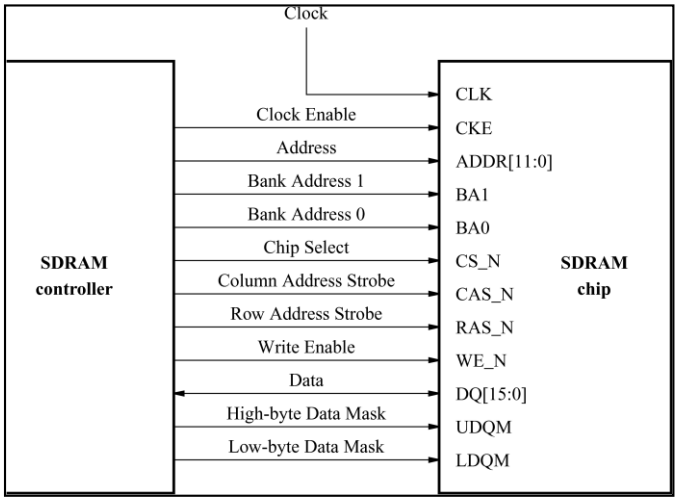
En nuestro caso no cumpliremos con esto, debido a que la SDRAM incrustada en la tarjeta DE0-nano es de 16 bits mientras que como sabemos el procesador Nios II es de 32 bits.

Uso de otras memorias

Muchas aplicaciones embebidas requieren tanto: memorias volátiles y no volátiles porque los dos tipos de memorias sirven para propósitos exclusivos.

SDRAM Controller

El SDRAM Controller provee todas las señales necesarias para comunicarse con el off-chip SDRAM excepto la señal de clock que debe satisfacer los requerimientos de la memoria en cuanto al desplazamiento del reloj, para una apropiada operación del chip SDRAM es necesario que la entrada de clock del chip adelante por 3 nanosegundos al reloj del sistema Nios II. Este reloj puede ser provisto por un PLL (Phase-Locked Loop) creado manualmente con la herramienta MegaWizard plug-in o creado automáticamente al usar el componente Clock Signals.



Señales del chip SDRAM y el cntrolador

Ajustes de tiempo del componente SDRAM Controller de Qsys

Del datasheet de la SDRAM ISSI IS42S16160B se extraen los valores de los parámetros de tiempo que se deben configurar en el controlador de la SDRAM.

Ajuste	Valor por defecto	Valor de la SDRAM IS42S16160B	Unidad	Descripción
CAS latency cycles	3	3	[ciclos]	Retardo entre el comando de lectura y la salida de datos. También puede configurarse con 2 ciclos
Initialization refresh cycles	2	8	[ciclos]	Cuantos ciclos de reloj debe realizar como parte de la secuencia de inicialización después de un reset.
Issue one refresh command every:	15.625	7.8125	[us]	Especifica que tan seguido el controlador refresca a la SDRAM. La IS42S16160B requiere 8192 comandos de refrescar cada 64 ms. Cada comando se da en 64ms / 8192 = 7.8125us
Delay after powerup before initialization	100	200	[us]	Tiempo de retardo para estabilizar el reloj luego del encendido y empezar proceso de inicialización de la SDRAM
t_rfc	70	70	[ns]	Periodo de auto refrescarse
t_rp	20	20	[ns]	Periodo de precargar
t_rcd	20	20	[ns]	Retardo desde el commando ACTIVE hasta los commandos READ o WRITE
t_ac	17	5.4	[ns]	Tiempo de acceso del flanco de reloj
t_wr	14	14	[ns]	Recuperación de escritura si un comando explícito de precarga es emitido

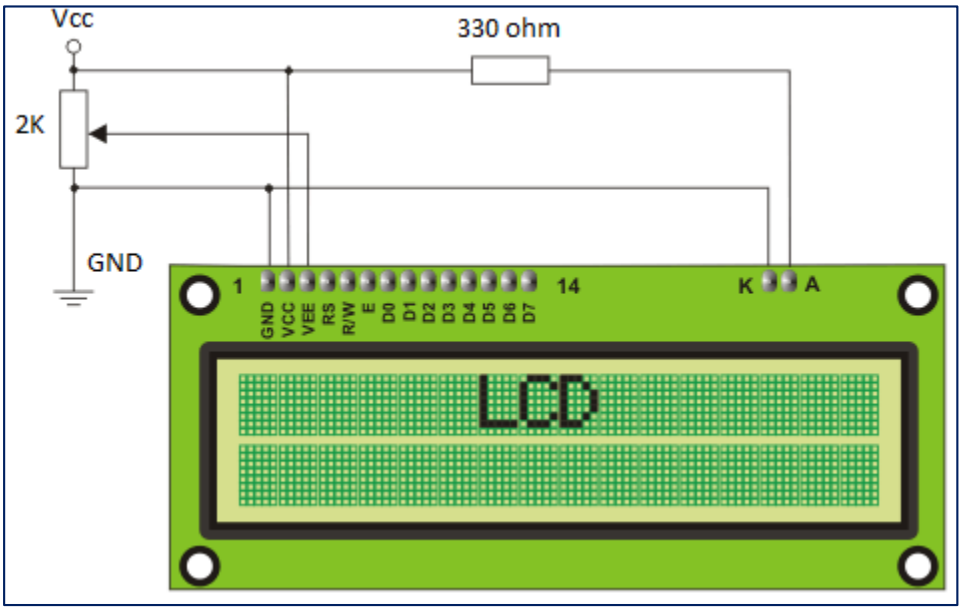
PANTALLA LCD

La pantalla LCD 16x2 cuenta con 16 pines (14 en caso de no tener el control de luz de fondo), puede mostrar dos líneas con 16 caracteres en cada una. El contraste de la pantalla depende del voltaje de alimentación y si los mensajes son presentados en una o dos líneas, por esta razón se aplica un voltaje entre 0 y el voltaje del pin llamado Vdd al pin de control de contraste llamado Vee.

Función	Número de Pin	Nombre	Estado lógico	Descripción
Tierra	1	Vss	-	0V
Alimentación	2	Vdd	-	+5V
Contraste	3	Vee	-	0 - +5V
Operación de control	4	RS	0 1	D0 – D7 son interpretados como comandos D0 – D7 son interpretados como datos
	5	R/W	0 1	Escribir datos (desde el controlador a la LCD) Leer datos (desde la LCD al controlador)
	6	E	0 1 Desde 1 a 0	Acceso a LCD deshabilitado Operación normal Datos o comandos son transferidos a la LCD
	7	D0	0/1	Bit 0 LSB
	8	D1	0/1	Bit 1
	9	D2	0/1	Bit 2

Datos / comandos	10	D3	0/1	Bit 3
	11	D4	0/1	Bit 4
	12	D5	0/1	Bit 5
	13	D6	0/1	Bit 6
	14	D7	0/1	Bit 7 MSB
Luz de fondo	15	K	-	0V
	16	A	-	0 - +5V

Pines de la pantalla LCD

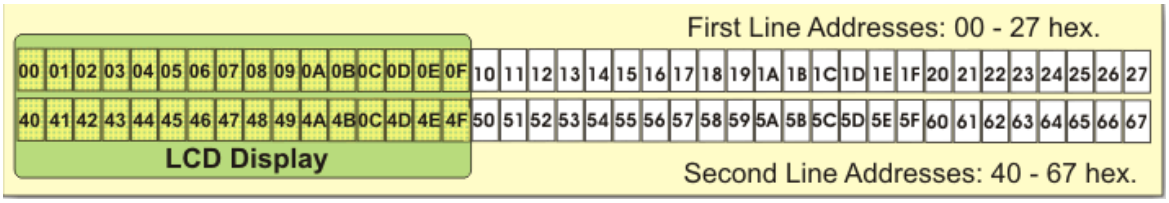


Conexión de los pines de alimentación, tierra, contraste y luz de fondo.

La pantalla LCD cuenta con tres bloques de memoria:

- **DDRAM (Display Data RAM).-** Es usada para almacenar los caracteres a ser mostrados. El tamaño de esta memoria es capaz de almacenar 80 caracteres.

Para el uso de la pantalla LCD 16x2, sólo se utilizaría desde la dirección en hexadecimal 00 hasta 0F para la primera línea y 40 hasta 4F para la segunda línea.



En caso de utilizar una pantalla LCD 20x4, se utilizaría desde la dirección en hexadecimal 00 hasta 13 para la primera línea, desde 40 hasta 53 para la segunda línea, a partir de 14 hasta la 27 y desde la 54 hasta la 67 para la cuarta línea.

- **CGROM.-** Contiene un mapa de caracteres estándares con todos los caracteres que pueden ser mostrados en la pantalla, cada carácter es asignado a una locación de memoria.

- **CGRAM.**- A parte de caracteres estándares, la LCD puede mostrar símbolos especiales definidos por el usuario, pueden ser símbolos definidos en el tamaño de 5 pixeles

Todo dato transferido a la LCD a través de las salidas D0 – D7 será interpretado como un comando o un dato dependiendo del estado de RS. Como se indicó en la tabla anterior si RS=1 entonces los bits D0 – D7 son direcciones de los caracteres a ser mostrados en la pantalla. La dirección DDRAM especifica la locación desde la cual serán mostrados los caracteres. Si RS=0 los bits D0 – D7 son comandos para ajustar el modo de la LCD.

El bit R/W será ajustado a cero desde código porque sólo se ejecutará el proceso de escritura, también se pudo haber ajustado a ese valor desde hardware.

COMANDO	RS	RW	D7	D6	D5	D4	D3	D2	D1	D0
Setear la dirección SDRAM	0	0	1	Dirección DDRAM						
Escribir a la CGRAM o DDRAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Limpiar la pantalla	0	0	0	0	0	0	0	0	0	1

Comandos reconocidos por la LCD y más utilizados

Fuentes de información:

1. DE0-Nano User Manual v1.9
2. Using the SDRAM on Altera's DE0-Nano Board with VHDL Designs
3. Memory System Design
4. 256-MBIT SYNCHRONOUS DRAM
5. Clock Signals for Altera DE Series
6. Book: PIC Microcontrollers-Programming in C (http://www.mikroe.com/chapters/view/17/chapter-4-examples/)

FUNDAMENTACIÓN TEÓRICA PRÁCTICA # 3

TEMA: *Adquisición de datos de los canales del chip convertidor ADC integrado en la tarjeta DE0-Nano.*

ADC128S022

Es el convertidor analógico-digital de bajo consumo de potencia que se encuentra integrado en la tarjeta DE0-Nano. Tiene ocho canales de entradas analógicas, cada una de las mismas se convierten a señales digitales de 12 bits. Su tasa de rendimiento esta entre 50 – 200 ksps (kilo samples per second); lo que quiere decir que puede tomar desde 50 hasta 200 muestras por segundo. La señal digital de salida utiliza el tipo de comunicación serial SPI que es soportado por la FPGA Cyclone IV. Entre las aplicaciones que se le pueden dar a este convertidor están: instrumentación médica, comunicaciones móviles e instrumentación y control de sistemas.

A continuación se muestra la distribución del cabezal de 2x13 pines de la tarjeta DE0-Nano, el mismo que provee acceso a las ocho entradas analógicas del ADC128S022 en los pines que se muestran en la *Figura 1*.

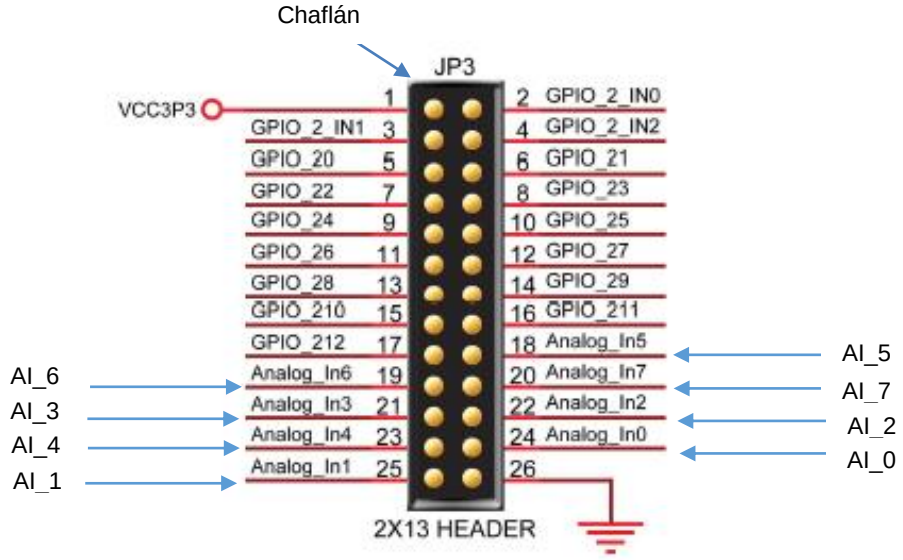


Figura 1. Ubicación de las entradas analógicas en el cabezal de pines 2x13 (vista superior)

Máxima entrada analógica.- Para evitar el daño de la tarjeta DE0-Nano cualquier voltaje provisto al ADC por medio del cabezal de pines de 2x13 no debe exceder los 3.3V.

En la tabla 1 se visualiza como se debería realizar las conexiones entre el chip ADC y la FPGA, en la columna tipo se indica si es entrada (I) o salida (O) con respecto a la FPGA. Ejemplo: En Quartus II se le dará el nombre ADC_SDAT, tipo entrada y se le asigna el pin_A9 estableciendo así la conexión con el pin DOUT del chip ADC, por supuesto desde el punto de vista del chip este pin es de salida.

Tipo	***Nombre del pin en el chip ADC***	***Nombre de la señal en la FPGA***	***No. Pin de la FPGA***	***Descripción de los pines del chip ADC128S022***
O	CS_n	ADC_CS_N	PIN_A10	*En los flancos de bajada de CS_n comienza la conversión y se mantiene mientras este abajo*
O	DIN	ADC_SADDR	PIN_B10	*Entrada de datos digitales. El registro de control del ADC se carga en los flancos de subida del SCLK*
I	DOUT	ADC_SDAT	PIN_A9	*Salida de datos digitales. Está sincronizado con los flancos de bajada del SCLK*
O	SCLK	ADC_SCLK	PIN_B14	*Entrada de clock digital. Para garantizar el desempeño la frecuencia debe estar entre 0.8 MHz a 3.2 MHz.*

Tabla 1. Asignación de pines para el ADC

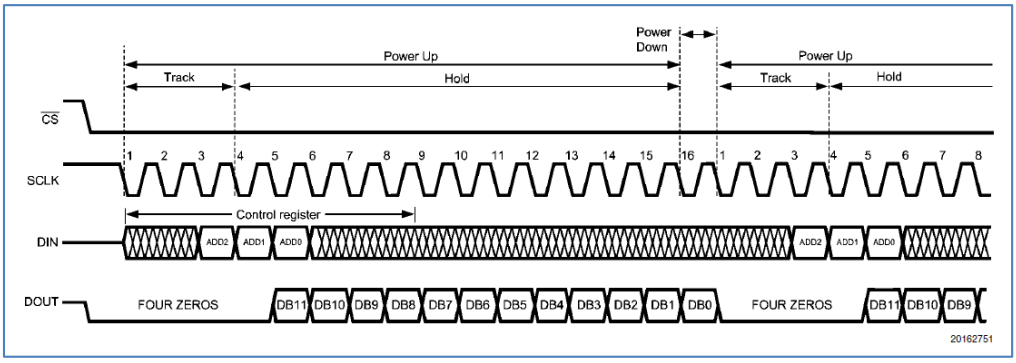


Figura 2. Diagrama de tiempo de operación del ADC128S022

Como se observa en la figura 2, para llevar a cabo la conversión de un dato analógico a un dato digital al chip ADC128S022 le toma 16 flancos de reloj del ADC (SCLK). El inicio de la conversión se da desde que la entrada $\overline{CS}$ recibe un flanco negativo y se mantiene así, se lee además el registro de control sobre el próximo canal a leer y envía por DOUT los 12 bits de salida que representan a la entrada analógica en binario. Con una frecuencia de 3.2MHz se obtendría (3.2M/16) Ksps; es decir con un reloj a esa frecuencia se toman 200000 muestras por segundo.

Por defecto, al iniciar el proceso de conversión, el convertidor toma el dato del canal 0 y para la siguiente conversión toma el dato del canal correspondiente de acuerdo a lo que se envíe por la entrada (con respecto al chip ADC) DIN, que recibe tres bits, desde 000 hasta 111 para representar los 8 canales disponibles.

El chip ADC es controlado por el componente *DE0-Nano ADC Controller,* el mismo que se encuentra en la sección *University Program* de la librería de componentes en Qsys (*en caso de no contar con esta librería véase Anexo Instalación University Program Library*). Este componente provee una interfaz de registro asignado en memoria cuya dirección se obtiene al hacer la asignación de direcciones base en Qsys.

Consideremos el caso particular de un Base Address = 0x00009000 y End Address = 0x0000901F, la distribución de la memoria en base a los ocho canales del ADC sería como se muestra en la tabla 2.

ADDRESS	31…12	11…0	Registro de los canales
0X00009000	Sin usar	Valor del canal 0	Registro del canal 0 / Update
0X00009004	Sin usar	Valor del canal 1	Registro del canal 1/ Auto-Update
0X00009008	Sin usar	Valor del canal 2	Registro del canal 2
0X0000900C	Sin usar	Valor del canal 3	Registro del canal 3
. .	. .	. .	. .
0X0000901C	Sin usar	Valor del canal 7	Registro del canal 7

Tabla 2. Distribución de memoria para el chip ADC

Para tomar las lecturas y convertirlas se debe escribir 1 en el bit 0 del registro del canal 1/ Update, a su vez se puede configurar el controlador para continuamente actualizar mediante la escritura en el bit 0 del registro del canal 1 / Auto-Update.

Se accede al valor adquirido más reciente de cada canal mediante la lectura del registro de dicho canal. La escritura sobre algún registro diferente al de Update o Auto-Update es ignorado.

Los niveles de tensión a conectar en las entradas del ADC no deben superar los 3.3V, es por ello que para la práctica se debe realizar un divisor de tensión con una alimentación de ese valor máximo de voltaje. En aplicaciones reales, los valores a medir son superiores, para ello deberá acondicionarse la señal mediante etapas de acoplamiento y amplificación.

Fuentes de información:

1. ADC128S022, 8-Channel, 50 kSPS to 200 kSPS, 12-Bit A/D Converter
2. DE0-Nano ADC Controller
3. DE0-Nano User Manual v1.9

FUNDAMENTACIÓN TEÓRICA PRÁCTICA # 4

TEMA: *Lectura y escritura sobre el chip ADXL345, acelerómetro de 3 ejes con hasta 13 bits de resolución, integrado en la tarjeta DE0-Nano.*

ADXL345

Es un pequeño acelerómetro de 3 ejes, con bajo consumo de energía y alta resolución de medida (hasta 13 bits). Se puede conectar a través de una interfaz de 3 cables SPI o de dos cables I2C. En la tarjeta DE0-Nano el acelerómetro está conectado a la FPGA mediante el arreglo SPI de 3 hilos. Entre sus aplicaciones se pueden mencionar: instrumentación médica e industrial, dispositivos de navegación personal, dispositivos señaladores, protección de discos duros, como sensor de inclinación permitiendo medir variaciones menores a 1°.

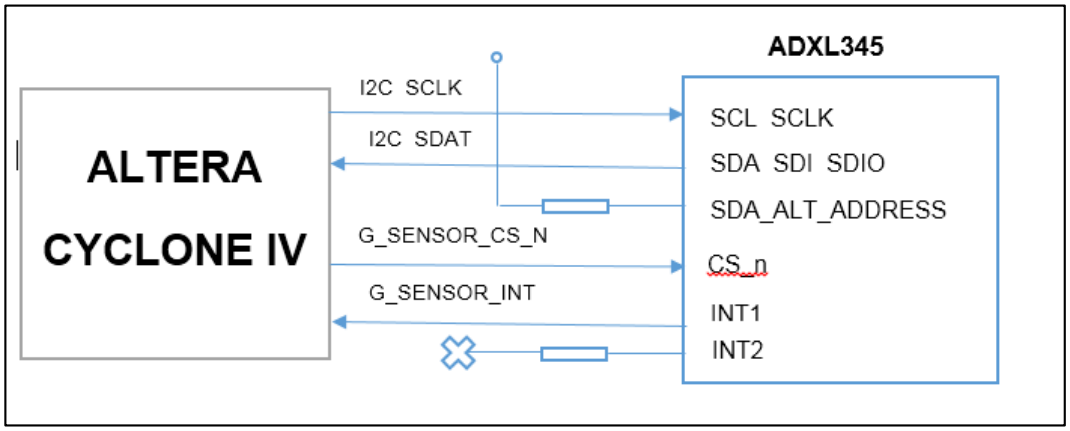


Figura 1. Conexiones entre el chip ADXL345 y la FPGA Cyclone IV

En la tabla 1 se visualiza como se debería realizar las conexiones entre el chip ADXL345 y la FPGA, en la columna tipo se indica si es entrada (I) o salida (O) con respecto a la FPGA. Ejemplo: En Quartus II se le dará el nombre I2C_SCLK, tipo salida y se le asigna el pin_F2 estableciendo así la conexión con el pin SCL_SCLK del chip ADXL345, por supuesto desde el punto de vista del chip este pin es de entrada.

Tipo	***Nombre del pin en el chip ADXL345***	***Nombre de la señal en la FPGA***	***No. Pin de la FPGA***	***Descripción de los pines del chip ADC128S022***
O	SCL_SCLK	I2C_SCLK	PIN_F2	
I	SDA_SDI_SDIO	I2C_SDAT	PIN_F1	
O	CS_n	G_SENSOR_CS_N	PIN_G5	*Selecciona el modo de comunicación.* $\overline{CS} = 1$ *para I2C y* $\overline{CS} = 0$ *para SPI*
I	INT1	G_SENSOR_INT	PIN_M2	

Tabla 1. Asignación de pines para el Acelerómetro

La resolución de salida depende la selección del rango g, para aplicaciones de medición de inclinación el rango óptimo es ±2 g.

Resolución de salida		**Cada eje**
Todos los rangos g	10 bits de resolución	10 bits
Rango ±2 g	Resolución full	10 bits
Rango ±4 g	Resolución full	11 bits
Rango ±8 g	Resolución full	12 bits
Rango ±16 g	Resolución full	13 bits

Tabla 2. Resoluciones de salida disponibles

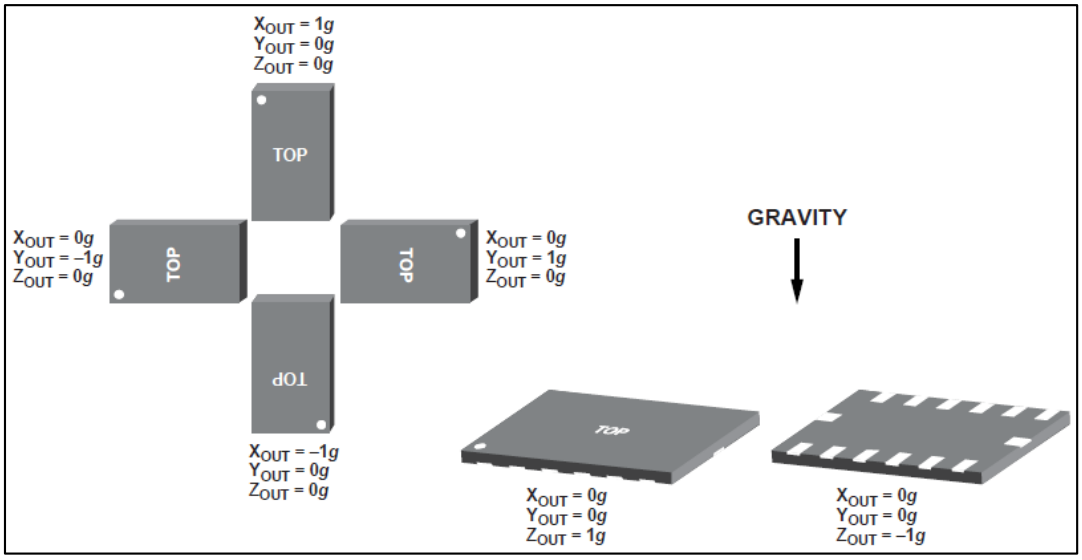


Figura 2. Respuesta de salida vs orientación respecto a la gravedad

El componente de Qsys, Accelerometer SPI mode, tiene dos registros de 8 bits, llamados address y data. Los mismos son mapeados a direcciones específicas una vez que al componente se le asigna las direcciones base en Qsys.

Por medio de estos registros el procesador puede especificar el modo de operación del acelerómetro y leer sus valores actuales.

OFFSET EN BYTES	NOMBRE DEL REGISTRO	LECTURA/ESCRITURA	7…6	5…0
0	address	E	0	Dirección
1	data	L/E	Dato del registro	

Para acceder a las medidas de los tres ejes se cuenta con dos pares de registros por eje, uno para los 8 bits más significativos (MSB) y otro para los LSB, para ello se debe escribir la dirección correspondiente en los 6 bits menos significativos del registro *address:*

EJE	DATO	Dirección para acceder
X	DATA X0 (8 bits menos significativos)	0X32
	DATA X1 (8 bits más significativos)	0X33
Y	DATA Y0 (8 bits menos significativos)	0X34
	DATA Y1 (8 bits más significativos)	0X35
Z	DATA Z0 (8 bits menos significativos)	0X36
	DATA Z1 (8 bits más significativos)	0X37

Cada vez que el diseño de hardware es cargado a la tarjeta, se realiza una auto inicialización del componente Accelerometer SPI mode, que tiene una configuración automática que causa el desarrollo continuo de las mediciones y permite la lectura de los resultados en las parejas de registros de X, Y y Z usando una resolución de ±2 g.

Los valores obtenidos se los puede presentar en decimal, simplemente almacenándolos en una variable de ese tipo; pero para obtener su valor de aceleración gravitacional se aplica la siguiente fórmula:

$$valor_{gravitacional} = \frac{valor_{decimal}}{511} * 2$$

Fuentes de información:

1. DE0-Nano User Manual v1.9
2. Accelerometer SPI Mode Core for DE-Series Boards
3. Digital Accelerometer ADXL345 Datasheet

FUNDAMENTACIÓN TEÓRICA PRÁCTICA # 5

TEMA: *Sistemas embebidos de arranque mediante el uso de la memoria flash Spansion S25FL064P, integrada en la tarjeta DE0-Nano y uso del Altera Monitor Program.*

Memorias no volátiles

A diferencia de las memorias volátiles como la SDRAM o la On-Chip memory, las memorias no volátiles retienen su contenido cuando se le quita la energía, lo que las convierte en una buena alternativa para el almacenamiento de información que debe ser recuperada después de desconectar y volver a conectar la alimentación de voltaje. El código de arranque de un procesador, los ajustes de aplicaciones frecuentes y los datos de configuración de la FPGA suelen ser típicamente almacenados en las memorias no volátiles.

La memoria flash es un tipo de memoria no volátil utilizada frecuentemente en sistemas embebidos. Este tipo de memoria es siempre externa en sistemas embebidos basados en FPGA porque las FPGA no contienen memoria flash. Las memorias flash están disponibles con interfaces en paralelo y en serie, para ambos casos la tecnología de almacenamiento es la misma.

La memoria flash presenta grandes ventajas sobre todo sus capacidad de no volatilidad, es durable, es borrable y permite un gran número de ciclos de borrado; pero su mayor desventaja es la velocidad de escritura sobre la misma que puede tomar cientos de ciclos de reloj, dependiendo de la velocidad del reloj.

La tarjeta DE0-Nano cuenta con una memoria flash Spansion EPCS64, este dispositivo es de configuración serial.

Componente Interval Timer

Este componente es un temporizador de intervalo para sistemas de procesadores basados en Avalon como Nios II, entre otras características, provee:

- Un contador de 32 y 64 bits
- El control para iniciar, detener y resetear el temporizador
- Dos modos de conteo: contar hacia atrás una vez o continuamente
- Registro del periodo de conteo hacia atrás
- Generador de pulsos opcional que entrega una salida de un pulso cuando alcanza cero.
- Compatible con procesadores de 32 y 16 bits

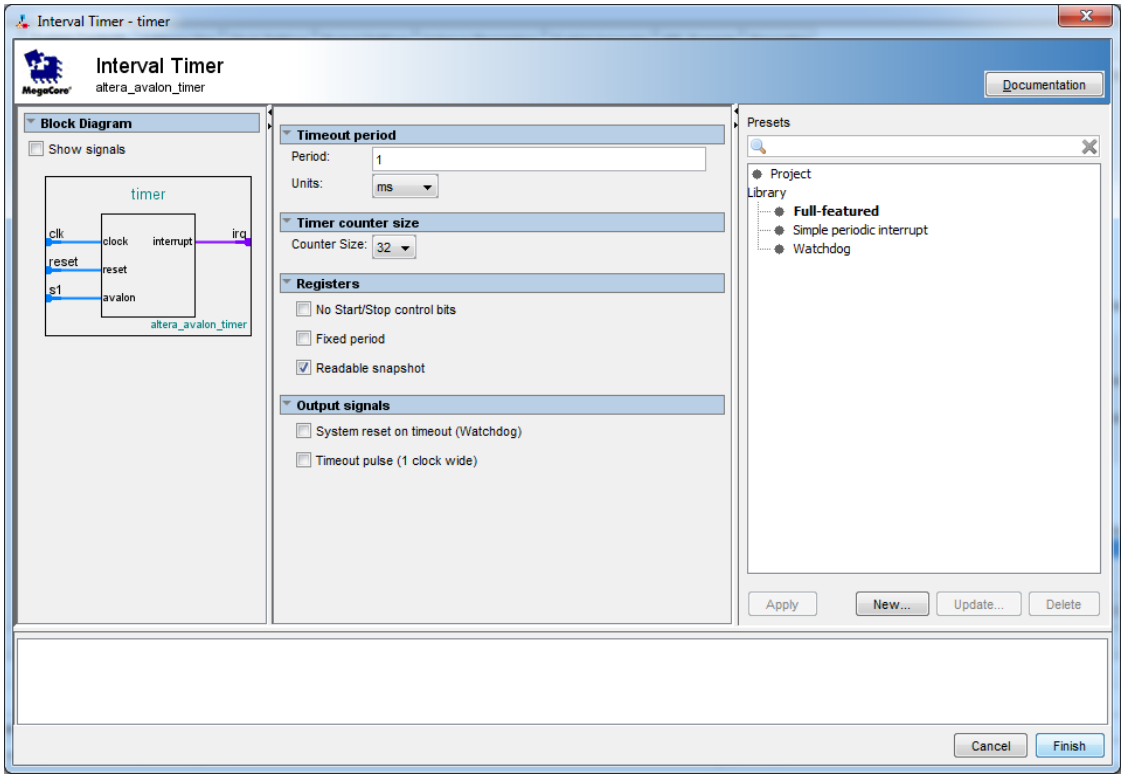
Offset	Name	R/W	Description of Bits						
			15	...	4	3	2	1	0
0	status	RW	(1)					RUN	TO
1	control	RW	(1)			STOP	START	CONT	ITO
2	periodl	RW	Timeout Period – 1 (bits [15:0])						
3	periodh	RW	Timeout Period – 1 (bits [31:16])						
4	snapl	RW	Counter Snapshot (bits [15:0])						
5	snaph	RW	Counter Snapshot (bits [31:16])						

Registros del Interval Timer de 32 bits

El periférico maestro Avalon-MM, como el procesador Nios II, puede escribir sobre el *registro de control* del componente para iniciar o detener el temporizador, habilitar interrupciones, especificar entre los modos de conteo, además puede escribir sobre los *registros de periodo (periodl, periodh)* para especificar el periodo del temporizador. El procesador también puede leer el valor actual del contador.

Ciertos registros sólo existen en hardware para ciertas configuraciones, como por ejemplo si en la configuración se seleccionara periodo fijo (*fixed period*) los registros de periodo no existiría en hardware y por lo tanto no se podría acceder a los mismos desde aplicaciones de software, por eso nosotros no seleccionaremos esa casilla en el desarrollo de la práctica porque sí accederemos a dichos registros. De manera similar verificaremos que se encuentre seleccionada la casilla de Readable snapshot

para que los registros que contienen el valor instantáneo del contador (snapl y snaph) existan en hardware.



Altera Monitor Program

Es una aplicación de software que corre sobre una computadora y se comunica con el sistema Nios II sobre una tarjeta FPGA que puede ser usado para compilar, ensamblar, descargar y depurar programas para el procesador Nios II de Altera.

Entre las características que presenta esta aplicación están:

- Inicializar un proyecto Nios II especificando un sistema de hardware y un programa de software.
- Descargar el sistema de hardware sobre la tarjeta FPGA
- Compilar los programas especificándolo en lenguaje ensamblador o C y descargar el resultante código de máquina en el sistema de hardware Nios II.
- Correr el procesador Nios II de corrido o por paso las instrucciones
- Examinar y modificar los contenidos de los registros Nios II

- Examinar y modificar los contenidos de memorias

Esta aplicación viene como parte de la instalación de la Suite de diseño del programa universitario de altera (*Altera University Program*)

Fuentes de información:

1. DE0-Nano User Manual v1.9
2. Memory System Design
3. Bootable Embedded Systems for the DE0-Nano Board
4. Embedded Peripherals IP User Guide (Interval Timer Core. Pag 326)
5. Altera Monitor Program Tutorial

FUNDAMENTACIÓN TEÓRICA PRÁCTICA # 6

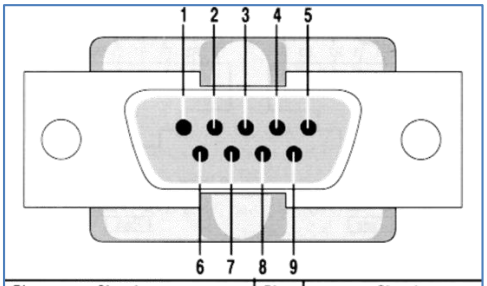
TEMA: *Comunicación serial RS-232 entre un sistema embebido sobre la FPGA de la tarjeta DE0-Nano y Proteus.*

Interfaz RS-232

RS-232 es un estándar para la transmisión de datos en una comunicación serial. Este tipo de comunicación se ve obstaculizada por una baja velocidad de transmisión, oscilaciones grandes de voltaje y conectores que en la actualidad se consideran de gran tamaño y por lo tanto han sido desplazados de los computadores modernos por el USB (*Universal Serial Bus*); pero se puede conectar a la misma a través de convertidores de RS-232 a USB, pese a lo mencionado aún se utilizan dispositivos RS232 y especialmente en maquinaria industrial, equipos de redes, balanzas electrónicas, registradores de datos (data-loggers).

La interfaz RS-232 es un enlace bidireccional punto a punto, con dos señales independientes para establecer la comunicación en los dos sentidos (tx, rx), además cuenta con señales adicionales para el control del flujo de datos. La norma básica utiliza 25 líneas (datos + control) y conectores DB-25; aunque es normal encontrar la versión de 9 pines (conocido como conector DB-9). La versión actual del estándar es TIA-232-F.

Niveles de tensión.- +3 a +12 voltios indica un estado lógico 0, mientras -3 a -12 voltios indica un estado lógico 1, entre -3 a +3 voltios es un rango de indeterminación.



Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

Con las señales de los pines 2 (recepción de datos), 3 (transmisión de datos) y 5 (señal de tierra) se puede establecer la comunicación serial y es como procederemos a realizarlo en la presente práctica.

Componente UART (RS-232 Serial Port)

Se encuentra en la librería de componentes de Qsys, cuenta con una interfaz Avalon-MM para que el procesador Nios II pueda comunicarse con éste mediante lectura y escritura en sus registros de control y datos e implementa un método para realizar el intercambio serial continuo de caracteres entre un sistema embebido en la FPGA de Altera y un dispositivo externo.

El componente cuenta con parámetros de configuración, los mismos que deben ser iguales a las que se establezcan en el otro extremo de la comunicación.

Ajuste	Valores posibles	Descripción
Paridad	None, Even, Odd	Determina si el componente UART transmite y recibe caracteres con chequeo de paridad o no.
Data Bits	7, 8, 9	Determina el ancho del dato de transmisión, recepción y registros endofpacket.
Stop Bits	1, 2	Determina si el componente transmite 1 o 2 bits de parada con cada caracter.

Synchronizer stages	2, 3, 4, 5	Permite especificar la longitud de cadenas de registros de sincronización, esta cadena es usada cuando ocurre un evento con disturbios y esta longitud determina el tiempo promedio antes del fallo.
Include CTS/RTS	ON (visto), OFF	En ON, se implementan los bits de control y estado CTS, DCTS, IDCTS y RTS
Include end-of-packet	ON (visto), OFF	Permite al UART terminar la transmisión de datos con el procesador mediante el control de flujo al identificar un carácter que se fija en el registro endofpacket.
Baud rate (bps)	Todas de acuerdo al estándar RS232.	Determina la velocidad de la comunicación expresada en bits por segundo.
Fixed baud rate	ON (visto), OFF	Cuando esta encendido se crea un registro de 16 bits (divisor), al escribir en el mismo se fija el valor de los bps según: $baud\ rate = \frac{clock\ frequency}{divisor + 1}$

Fuentes de información:

1. Embedded Peripherials IP User Guide (UART Core)
2. https://en.wikipedia.org/wiki/RS-232
3. http://www.arcelect.com/rs232.htm

Anexo 3. Creación de un proyecto en Quartus II

1. Luego de abrir Quartus II, seleccionar la opción **Create a New Project (New Project Wizard).**

Figura 1. Selección de crear un nuevo Project Wizard

2. Se abre una ventana de introducción, clic en **Next**. En la primera página de la creación del proyecto que se presenta, seleccionar una carpeta de directorio, en la misma se almacenará todos los archivos generados por Quartus II y los que generen sus herramientas como Qsys, Nios II Software Build Tools for Eclipse, etc (*Véase Figura 2*).

3. En la misma primera página escribir un nombre para el proyecto y para el archivo de mayor rango en la compilación, por defecto los dos tendrán el mismo nombre (*Véase Figura 2)*. Clic en **Next**. Y en la segunda página no se añadirá ningún archivo, por ello clic en **Next**

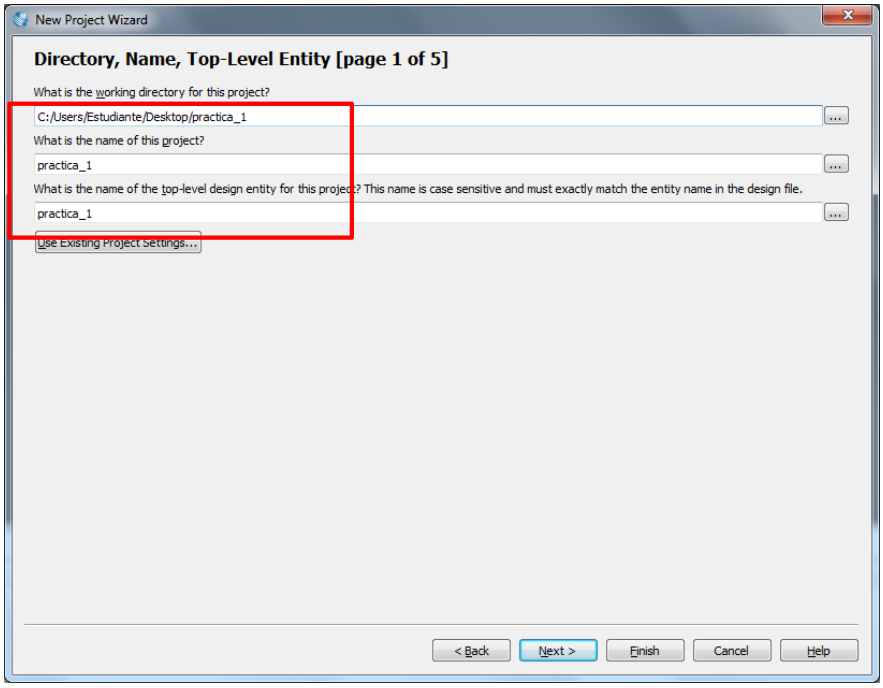


Figura 2. Selección de un directorio, nombre del proyecto y nombre de la entidad de mayor nivel

4. En la tercera página de la creación de un proyecto se procede a añadir la familia y dispositivo de Altera con el que se trabajará. Clic en **Next**.

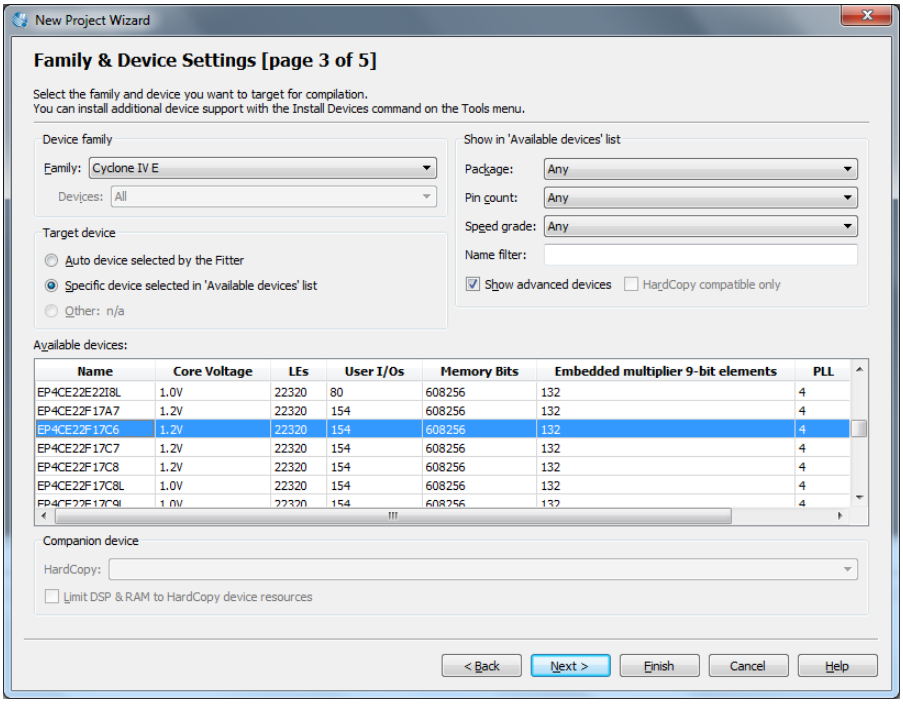


Figura 3. Selección de la familia y dispositivo de la FPGA con el que se cuenta en la tarjeta de desarrollo DE0-Nano.

5. En la cuarta página de la creación del proyecto se muestran opciones de simulación, en caso de no utilizar, se selecciona **None** en todas las opciones. Clic en **Next**.

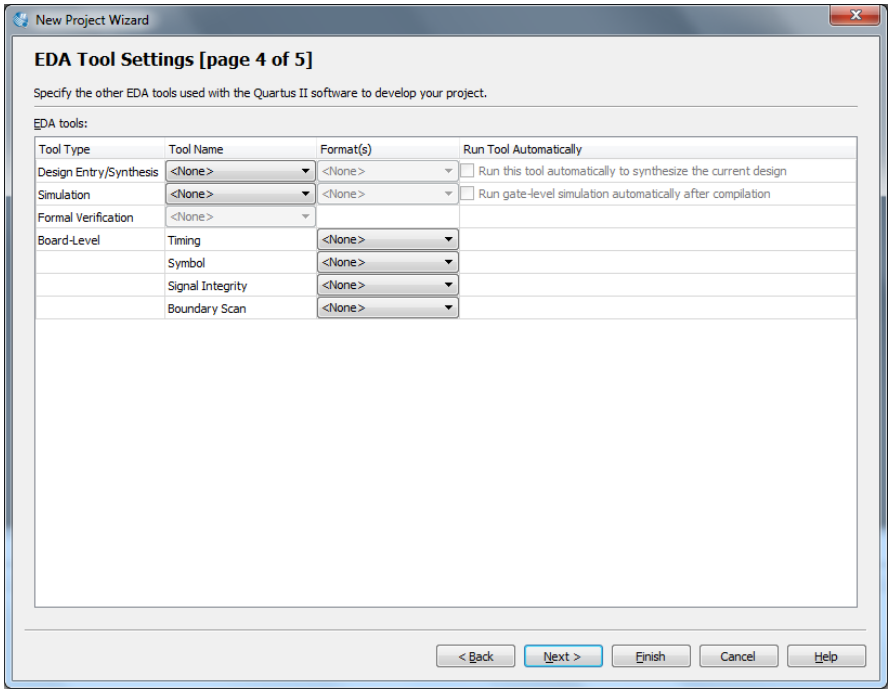


Figura 4. Ajustes de las herramientas EDA.

6. En la quinta página se muestra un resumen del proyecto creado. Clic en **Finish,**

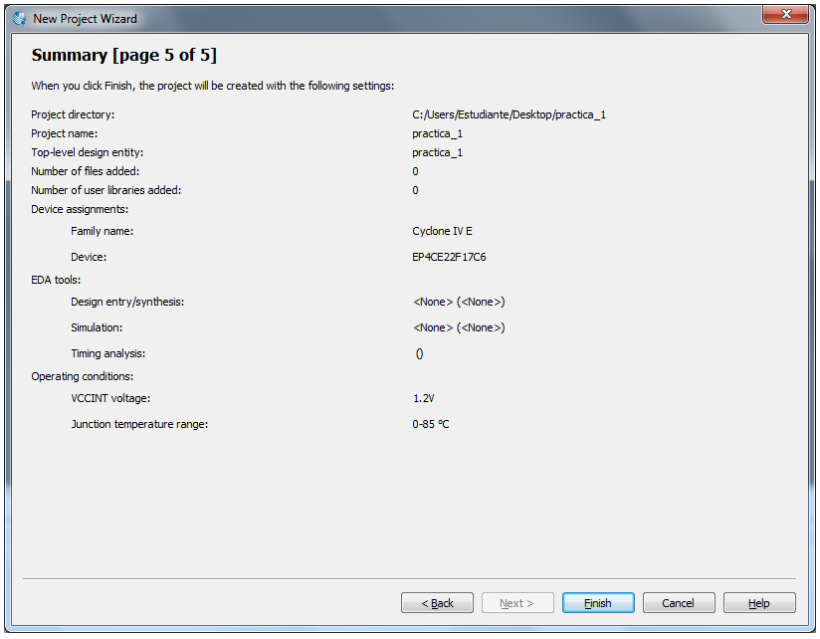


Figura 5. Resumen de la creación del proyecto en Quartus II