DISEÑO Y CONSTRUCCIÓN DE UN INSTRUMENTO PROTOTIPO PARA EL ANÁLISIS DE LA LEY DE HOOKE, CAIDA LIBRE Y PENDULO SIMPLE USANDO DISPOSITIVOS FPGAs

Figueroa Agudelo Carlos Andrés, ✉ cafigueroa@utp.edu.co
Giraldo Salazar Sergio Oswaldo, ✉ serosgisa@utp.edu.co

Tesis de Maestría presentada para optar al título de Magister en Instrumentación Física

Asesor: Hugo Armando Gallego, Magíster (MSc) en Física

Universidad Tecnológica de Pereira
Facultad de Ciencias Básicas
Maestría en Instrumentación Física
Pereira, Risaralda
2019

Cohorte III.

Grupo de Investigación: Docencia para la Investigación.

Línea de investigación: Diseño y Construcción de Equipo Didáctico para la Enseñanza de la Física.

Dedicatoria

A Dios, por permitirme alcanzar este nuevo escalón.
A Nicolás y Marisol, por ser mi motor y el impulso para continuar día a día.
A mis padres Horacio y Nelly por mostrarme el camino hacia la superación.
A mis hermanos Jorge Eduardo, Ana Mercedes, Esteban, por alentarme a seguir avanzando.

Sergio Oswaldo

A Dios, dador y cuidador de nuestra vida.
A mis padres Ramiro y Gloria por su constante apoyo.

Carlos Andrés

Agradecimientos

Varias personas contribuyeron en el proceso y conclusión de este trabajo, especialmente al Msc. Hugo Armando Gallego Becerra, profesor de la Facultad de Ciencias Básicas y director de este proyecto por estar siempre presto a colaborarnos a pesar de las múltiples dificultades que se presentaron.

Agradecimientos especiales al ingeniero Jerry Colorado, instructor de TecnoParque nodo Pereira quien facilitó varios elementos tipo hardware para dar concepción al sistema. Igualmente al Msc. Diego Fernando Salazar, profesor contratista de la Universidad Tecnológica de Pereira por su orientación en el manejo del SoftCore del sistema desarrollado.

En general, a todos los profesores de la Universidad Tecnológica de Pereira que impartieron sus conocimientos en la parte académica. A la asistente Administrativa de la maestría Karina Sánchez Sánchez por su colaboración ante el comité curricular en los diversos trámites y documentos para la etapa final.

En la Universidad del Valle, agradecimientos especiales los profesores PhD Jaime Velasco Medina y Msc Mario Enrique Vera Lizcano del grupo de Nanoelectrónica por sus aportes y sugerencias en la concepción del sistema.

Tabla de contenido

Lista de tablas

Lista de figuras

Glosario

BSP Board Support Package.

CPU Central Processing Unit.

HDL Hardware Description Language.

IDE Integrated Development Environment.

IP Intellectual Property.

JTAG Joint Test Action Group.

LED Light Emitting Diode.

PIO Parallel I/O.

SVN Subversion.

xme/chromosome a modular middleware architecture for cyber-physical systems.

ASIC Application Specific Integrated Circuit

BRAM Block RAM

CLB Configurable Logic Block

DCM Digital Clock Manager

DDR Dual Data Rate

DLL Delay Lock Loop

DSP Digital Signal Processing

EDA Electronics Design Automation

E/S Entrada – Salida

FIFO First Input First Output

FPGA Field Programmable Gate Array

HSTL High Speed Transceiver Logic

IP Intellectual Property

I/O IO Input –Output

ISP In-Circuit Programmable

IOB Input Output Block

JTAG Joint Test Action group

HMI Human Machine Interface

IRED Diodo Emisor de Infrarrojos

LVDS low-voltage differential signaling

LVTTL low-voltage TTL

Resumen

Este proyecto tiene como propósito diseñar y construir un prototipo que permita desarrollar una práctica de laboratorio para la comprobación de una ley y un principio de la Física, usando la lógica programable sintetizada en dispositivos como son los CPLDs (Complex Programmable Logic Device) o FPGAs (Field Programmable Gate Array), los cuales serán utilizados como elemento central en la construcción del mismo.

Se involucran las FPGAs (Field Programmable Gate Array), ya que éstas permiten realizar nuevas mediciones de variables Físicas y al mismo tiempo ofrecen la posibilidad de generar nuevo conocimiento y nuevas propuestas de investigación en este campo, empezando en la academia como generadora de modelos y posibles soluciones aplicables al sector productivo.

Para llevar a cabo la propuesta fue necesario realizar un estudio detallado sobre estos dispositivos, comenzando desde sus características técnicas en cuanto a hardware y programación, hasta la conexión con otros dispositivos (PC, sensores, visualizadores, motores, etc.) para su posterior calibración como se hace con cualquier instrumento de medición.

Como producto principal de este trabajo se elaboran las guías para desarrollo y comprobación de las leyes de la cinemática como son Péndulo simple, Caída Libre y Ley de Hooke.

Palabras clave: Cinemática, CPLD, FPGA, Instrumentación, Laboratorios, Leyes Físicas.

Abstract

The purpose of this project is to design and build a prototype that allows the development of a laboratory practice for the verification of a law and a principle of physics, using the programmable logic synthesized in devices such as CPLDs (Complex Programmable Logic Device) or FPGAs (Field Programmable Gate Array), which will be used as a central element in the construction thereof.

It is intended to involve FPGAs (Field Programmable Gate Array), since these allow to make new measurements of physical variables and at the same time offer the possibility of generating new knowledge and new research proposals in this field, starting at the academy as model generator and possible solutions applicable to the productive sector.

To carry out the proposal it is necessary to carry out a detailed study on these devices, starting from their technical characteristics in terms of hardware and programming, to the connection with other devices (PC, sensors, displays, motors, etc.) for their subsequent calibration as it is done with any measuring instrument.

As the main product of this work, the guidelines for the development and verification of the laws of kinematics are prepared, such as Simple Pendulum, Free Fall and Hooke's Law.

Keywords: CPLD, FPGA, Instrumentation, Kinematics, Laboratories, Physical Laws.

.

Introducción

Actualmente, en la Instrumentación para la Docencia, existen muchos instrumentos que miden determinadas variables físicas; dichos instrumentos basados en dispositivos como microcontroladores resultan ser muy eficaces por su fácil manejo, además que su valor agregado está en su costo lo que hace viable la construcción de instrumentos de medición basados en estos dispositivos con altos niveles de confiabilidad. Pero existen otras alternativas para el diseño e implementación de aparatos de medida, en lo que se refiere a dispositivos que procesan la información, como son los módulos PLD y FPGA, cada vez que la electrónica con estos dispositivos tiende a ser programada, es decir, bloques modulares facilitan su programación para determinadas tareas entre ellas fundamentalmente la recepción y transmisión de señales análogas y/o digitales. Como dichos dispositivos poseen un mayor nivel de integración es posible, además, con un solo dispositivo de determinadas características manejar procesos simples o complejos que involucran cálculos aritméticos o lógicos.

La implementación de instrumentos de laboratorio adoptando nuevas tecnologías, obligatoriamente involucra la necesidad de generar nuevo conocimiento y alternamente adopta la posibilidad de innovar y adaptar la nueva tecnología al proceso de la enseñanza en sus diferentes niveles, comenzando en la educación media, superior y de postgrado. Por estas razones, se pretende justificar la utilización de estos dispositivos para la realización de este tipo de proyectos, ya que cuentan con buena documentación.

Precisamente como no existen en el medio dispositivos de campo que permitan determinar o monitorear variables de procesos físicos o químicos que utilicen PLD o FPGAs, este proyecto contribuye no solamente a la solución del problema planteado, sino que se convierte en un abrebocas para futuros desarrollos en el campo de la docencia con fácil adaptación en el sector industrial si así lo requiere; y que la tecnología de la lógica programada permite fácilmente adaptarlo a determinado proceso.

Adicionalmente, se busca brindar una visión a las tecnologías y metodologías de diseño en el procesamiento digital de señales, con el fin de presentar aplicaciones prácticas sobre estas herramientas de hardware y software que estimulen la investigación y tener un contacto permanente con tecnología de punta.

Este proyecto se encamina a fomentar el desarrollo de proyectos con esta tecnología enfocado a aplicaciones prácticas en el procesamiento de señales.

CAPÍTULO 1
ASPECTOS GENERALES

1.1. Planteamiento del problema

Cuando se habla del cálculo y la determinación o comprobación de las variables físicas, para estudiantes de programas de pregrado, o incluso de educación media. Se deben incorporar las nuevas tecnologías, para éstas se necesita disponer de dispositivos versátiles, que a su vez manejen una programación sencilla de entender, que pueda adecuarse fácilmente, además que los valores obtenidos sean muy reales y precisos, todo esto enfocado a que los estudiantes de una manera fácil puedan apreciar y realizar las mediciones respectivas.

Por esto, se pretende utilizar un dispositivo como es la tarjeta de desarrollo Cyclone II 2C35, como paso inicial en la realización del primero de varios proyectos, debido al auge que tiene en la actualidad, a la buena documentación que se tiene acceso de esta tecnología y a las facilidades que entregan.

Adicionalmente, se busca brindar una visión a las tecnologías y metodologías de diseño en el procesamiento digital de señales, con el fin de presentar aplicaciones prácticas sobre estas herramientas de hardware y software que estimulen la investigación y tener un contacto permanente con tecnología de punta.

Este proyecto se pretende mostrar como el pionero que permita lograr el desarrollo y generar un trabajo en investigación con esta tecnología, enfocado a aplicaciones prácticas en el procesamiento de señales, caso especial la comprobación de muchos de los fenómenos físicos que a diario son estudiados.

1.2. Antecedentes

Para generar industria tecnológica en una economía en crisis, se deben crear productos aplicados que tengan la facilidad de utilización global sin tener los costos de un producto completamente especializado, esto se logra creando prototipos genéricos que se pueden utilizar en varias aplicaciones. En la última década el diseño y desarrollo de sistemas electrónicos digitales ha cambiado drásticamente, de ser un área casi exclusiva de la Ingeniería Electrónica ha pasado a ser un área que involucra diversas metodologías de diseño.

Este cambio ha hecho posible que el diseño de hardware sea casi tan flexible y conveniente como lo es el diseño de software y ha sido posible gracias a los dispositivos programables, de los cuales los más recientes y avanzados son los FPGA (Field Programmable Gate Array) (BROWN & VRANESIC, 2005) [4]. Los FPGA son un arreglo de bloques lógicos programables colocados en un arreglo programable de interconexiones, así es posible programar la función de los bloques lógicos, las interconexiones entre éstos y las conexiones de entradas y salidas. Estas características hacen que los costos de diseño y desarrollo de sistemas digitales se reduzca considerablemente, a la vez que el tiempo requerido para diseñar circuitos es mucho menor que con los métodos tradicionales.

Estos dispositivos tienen una característica llamada reconfiguración dinámica la cual básicamente permite cambiar las conexiones del circuito sin necesidad de desconectar el circuito o detener el proceso. Esta característica hace que estos dispositivos puedan adaptarse a cambios del sistema físico, reducir el consumo de potencia desconectando circuitos que no se estén utilizando en un momento dado, realizar diversas funciones lógicas con el mismo circuito o bien contar con un mecanismo de detección y corrección de fallas.

Para programar las FPGA se utilizan lenguajes descriptivos del tipo HDL (VHDL, Verilog) (Anon, 1995) [3], lo cual hace posible aplicar metodologías descendentes (top-down), es decir, realizar primero el diseño y preocuparse al final por las características del circuito electrónico. También es posible realizar co-diseño hardware-software, sistemas de cómputo que

no utilicen microprocesadores o hasta el diseñar sistemas hardware con la flexibilidad y potencial de los sistemas software.

Con base en lo expuesto anteriormente, el uso de dichas metodologías en el diseño e implementación de hardware utilizando dispositivos programables tales como los FPGA abre múltiples áreas de investigación y desarrollo tecnológico que, actualmente son aprovechadas solo en algunos países.

Actualmente existen algunos trabajos a nivel internacional relacionados con esta propuesta tales como, utilización de algoritmos genéticos y algoritmos de vida artificial para el diseño óptimo de circuitos lógicos y sistemas hardware adaptables utilizando métodos de evolución. Sin embargo a nivel nacional, y se podría decir latinoamericano, es necesario utilizar esta tecnología por todas las ventajas que presenta, así mismo es necesario formar recursos humanos especializados en esta área (WAKERLY, 1994) [10].

Los antecedentes son las investigaciones que se han realizado previamente y que guardan una relación histórica con el tema de investigación actual.

1.3. Justificación

En el presente documento se realiza el estudio y diseño de un Sistema Digital, basado en un procesador embebido en un FPGA, que constituye el elemento central de procesamiento de señales acondicionadas provenientes de diferentes tipos de sensores usados en el modelo. El sistema diseñado debe ser capaz de recibir señales usando diferentes protocolos de comunicación, tales como: Serial RS-232 y vía convertidor A/D (para el ingreso de información analógica al sistema).

Se hace uso de la metodología SoC (*System on Chip*), la cual es una creciente metodología de diseño en VLSI. Un tipo de SoC particular es el SoPC (*System on a*

Programmable Chip), el cual es más flexible y apropiado para pequeños lotes de producción, ya que combina las ventajas del diseño SoC y PLD (Dispositivos de Lógica Programable).

El sistema desarrollado tiene múltiples funciones ya que se programa dentro de la FPGA un procesador que se encarga de conectar el hardware adicional a la misma. Se plantea una primera estrategia de desarrollo y quedan abiertas otras posibilidades de desarrollo dentro del campo de instrumentos de laboratorio para Física; adicionalmente, este proyecto constituye una base para futuros desarrollos en el diseño de sistemas digitales basados en procesadores Soft-Core.

Se pueden usar diferentes interfaces de comunicación dependiendo de las necesidades del experimento; Los fabricantes ofrecen sistemas de desarrollo dentro de los programas educativos que resultan muy efectivos y que terminan en una buena alternativa a la hora de programarlos si se hace una comparación con dispositivos embebidos como los microcontroladores.

Conjuntamente el documento presenta las guías de desarrollo de las prácticas alusivas a los fenómenos físicos que se quieren comprobar.

1.4. Objetivo General

- Desarrollar un instrumento prototipo para comprobar experimentalmente la Ley de Hooke, Caída Libre y Péndulo Simple soportadas en sistemas de tipo hardware gracias a su alta capacidad de procesamiento.

1.5. Objetivos específicos

- Estudiar los conceptos básicos y las aplicaciones de los dispositivos FPGAs de un fabricante específico.

- Implementar algoritmos para el procesamiento de señales mediante Procesador Soft-Core en FPGAs.

- Utilizar los dispositivos FPGAs como elementos de procesamiento de información en instrumentos de medición de variables físicas.

- Realizar un instrumento prototipo de laboratorio basado en esta tecnología, para la comprobación de la *LEY DE HOOKE, CAIDA LIBRE Y PENDULO SIMPLE.*

- Elaborar una nueva guía para la experimentación de la Ley de Hooke, Caída Libre y Péndulo Simple, estableciendo comparaciones con los resultados arrojados por las prácticas tradicionales en los laboratorios de Física.

CAPÍTULO 2
MARCO TEÓRICO

2.1. Diseño de Sistemas basados en FPGA's

Actualmente el ingeniero desarrollador cuenta con varias herramientas Software y Hardware para poder trabajar cada uno de sus nuevos diseños, con diferentes opciones de dispositivos hardware para realizar Diseño Electrónico Digital, tales como: FPGA's, CPLD's, DSP's, uCo, entre otros. Cada uno de los cuales deberá elegir según las circunstancias y rigurosidades del problema que se presente.

Además de los sistemas hardware con los que cuenta, en la actualidad se tiene que:

- La realidad del diseño lógico actual exige el uso de herramientas de diseño electrónico del tipo EDA.
- En cada herramienta EDA se tiene asociada una metodología de diseño.
- Herramientas EDA con soporte de dispositivos del fabricante o independiente del fabricante.
- Posibilidad de diseñar sistemas completos en chip programable (*SoPC = System on Programmable Chip*).

Los Sistemas embebidos están compuestos por hardware y software que trabajan juntos para realizar una función específica. Usualmente ellos contienen procesadores embebidos que frecuentemente están en la forma de procesadores Soft-Core que ejecutan código de software.

Para ver de una manera más general a los procesadores, enumeramos lo siguiente:

a. Los núcleos de procesadores se pueden implementar en hardware o software.
b. Procesadores en hardware (*Hard-Core*): Usan un núcleo de procesador embebido en silicio dedicado y ofrecen ventajas híbridas entre FPGA y ASIC.

c. Procesadores en software (*Soft-Core*): Usan elementos lógicos programables existentes en el FPGA para implementar la lógica del procesador (Aberga Farro, 2014) [1].

2.2. Tendencias en la Implementación de Sistemas Digitales

Las alternativas para implementar un sistema digital, considerando su flexibilidad (programabilidad) y eficiencia (adaptación a la aplicación, en cuanto a velocidad, consumo de potencia, entre otros) se puede agrupar en tres categorías:

- Circuitos integrados estándar y ASIC.
- Microprocesadores, microcontroladores.
- Lógica programable.

Existen varias alternativas sobre el tipo de circuito integrado a utilizar para la implementación de un sistema digital y la elección de una de ellas dependerá, entre otras, de factores tales como la velocidad, el consumo de potencia, el costo y el volumen de producción. El diseño con circuitos integrados estándar y ASIC resulta eficiente y rápido. Sin embargo, carece de flexibilidad ya que no pueden alterarse una vez fabricados. Por otro lado, la producción en serie lleva a unos costos por chip muy bajos, además que la fase de producción de test garantiza la alta fiabilidad de estos circuitos. La tecnología actual permite crear circuitos integrados de gran complejidad y precio reducido, pero el costo de desarrollo de un ASIC solo se justifica en el caso de altos volúmenes de producción.

Los microprocesadores y microcontroladores son circuitos programables de arquitectura fija. Son menos eficientes que los circuitos a medida, la ventaja con ellos es que se tiene mayor flexibilidad, debido a que la variación de la función se consigue solo al modificar el software. Aunque los primeros microprocesadores eran demasiado lentos para implementar en tiempo real la mayoría de los sistemas, a mediados de los ochenta, la tecnología de los circuitos integrados había avanzado hasta permitir la realización de microcomputadores en punto fijo y punto flotante con arquitecturas especialmente diseñadas para realizar algoritmos de procesamiento de señales en tiempo discreto. A estos se les conoce con el nombre de DSP (Digital Signal Processor).

Algunas de sus características básicas como el formato aritmético, la velocidad, la organización de la memoria o la arquitectura interna hacen que sean adecuados para ciertas aplicaciones particulares, así como en otras más genéricas (Castillo Morales, 2008) [5].

2.3. Dispositivos Lógicos Programables (PLD)

Son circuitos integrados digitales normalizados que se caracterizan porque la función que realizan se puede cambiar mediante la programación de las conexiones entre los elementos que lo componen, bien sea secuenciales o combinatorios. Estos elementos son Flip-Flops, Latch y las Compuertas Básicas (AND, OR y NOT).

Los principales circuitos que pertenecen a los PLDs son:

- ROMs (Read Only Memory)
- PLAs (Programmable Logic Array)
- Pals (Programmable Array Logic)
- CPLDs (Complex Programmable Logic Device)
- FPGAs (Field Programmble Gate Array)
- ASICs (Application Specific Integrated Circuit)

Cuando se aborda el diseño de un sistema electrónico y surge la necesidad de implementar una parte con hardware dedicado son varias las posibilidades que hay. En la figura 1 se han representado las principales aproximaciones ordenándolas en función de los parámetros costo, flexibilidad, prestaciones y complejidad. Como se puede ver, las mejores prestaciones las proporciona un diseño full-custom, consiguiéndose a costa de elevados costos y enorme complejidad de diseño. En el otro extremo del abanico de posibilidades se encuentra la implementación software, que es muy barata y flexible, pero que en determinados casos no es válida para alcanzar un nivel de prestaciones relativamente alto.

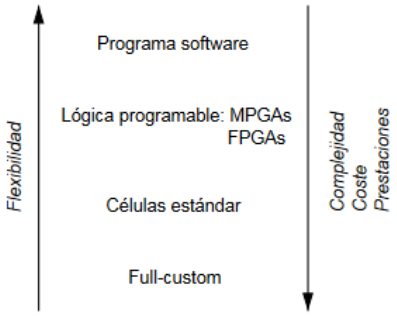


Figura 1. Diferentes soluciones para el diseño de circuitos digitales

https://books.google.com.co/books?id=wzYuOF6HFX0C&pg=PA22&lpg=PA22&dq=EP2C35+altera&source=bl&ots=X9EwqDGPkB&sig=ACfU3U3kozU4dpaaV8y30NueC7IN-gmkBg&hl=es-419&sa=X&ved=2ahUKEwilyOOg4sLkAhVtvFkKHXa5DFsQ6AEwDnoECAkQAQ#v=onepage&q=EP2C35%20altera&f=false

Los dispositivos FPGA son circuitos ya fabricados que se pueden "programar" in situ y por ende constituyen una buena opción para realizar diseños electrónicos digitales con una buena relación costo-beneficio.

2.4. Arreglo de Compuertas Programable por Campos (FPGAs)

La creación de este dispositivo deriva de los Circuitos Complejos de Lógica Programable (CPLD) cerca de los años 80's. Su creador fue unos de los fundadores de Xilinx (empresa dedicada al desarrollo e investigación de los FPGAs), Ross Freeman, el cual la desarrolló en 1985. Los FPGAs se ubican en el más alto nivel de dispositivos reprogramables de silicio. Usando bloques de lógica configurables y recursos de enrutamiento programable, estos dispositivos se pueden configurar de acuerdo a las necesidades requeridas por el usuario.

Los FPGAs también son completamente reconfigurables y se cambian instantáneamente adaptando un nuevo desempeño cuando se vuelve a compilar una configuración diferente de circuitería. En el pasado, la tecnología FPGA estuvo sólo disponible a ingenieros con una comprensión profunda de diseño digital de hardware. El crecimiento de nuevas herramientas de alto nivel de diseño, permite cambiar las reglas de programación de FPGA, con nuevas

tecnologías que convierten diagramas de bloques gráficos o incluso código C en circuitería digital de hardware.

Un FPGA contiene una matriz de celdas lógicas idénticas con interconexiones programables, esta matriz también se denomina Bloques Lógicos Configurables (CLB), esta matriz está rodeada por un anillo de bloques de interfaz entrada-salida. Estos bloques conectan las señales de los CLB, el cual contiene generadores de funciones, biestables y multiplexores para rutear las señales dentro de ellos.

A diferencia de los procesadores, los FPGAs llevan a cabo diferentes operaciones de manera paralela, por lo que éstas no necesitan competir por los mismos recursos. Cada tarea de procesos independientes se asigna a una sección dedicada del chip, y puede ejecutarse de manera autónoma sin ser afectada por otros bloques de lógica. Como resultado, el rendimiento de una parte de la aplicación no se ve afectado cuando se agregan otros procesos.

2.5. Beneficios de la tecnología FPGA

Un FPGA se convierte en un dispositivo por medio del cual puede desarrollarse casi cualquier circuito digital, y al ser programable se reducen sensiblemente los costos de desarrollo comparados con los de un ASIC. Además en aplicaciones de procesamiento de señales, las FPGA alcanzan mayor rendimiento que los procesadores digitales de señal (DSPs) debido al alto grado de paralelismo que poseen los algoritmos empleados lo que hace ideales a los FPGA para este tipo de aplicaciones. La arquitectura regular y flexible de un FPGA permite el procesamiento paralelo en aplicaciones que requieren altas velocidades de procesamiento.

2.6. Arquitectura general de un FPGA

Los FPGA's (*Field Programmable Gate Arrays*) son dispositivos semiconductores programables que están basados por una matriz de bloques lógicos configurables (CLB's) conectados vía interconexiones programables. A diferencia de los Circuitos Integrados de Aplicación Específica (ASIC's) donde el dispositivo es configurado para el diseño en particular,

los FPGA's pueden ser configurados para una aplicación deseada o según los requerimientos de funcionalidad.

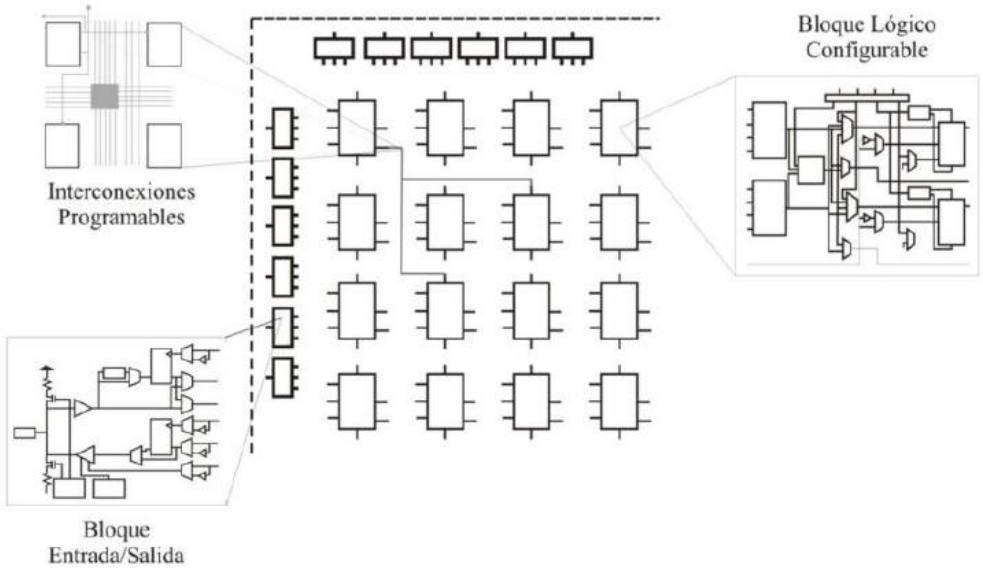


Figura 2. Elementos básicos en la arquitectura de un FPGA. (Ruiz Carbajal, 2012)[24]

En la Figura 2 se muestra la arquitectura básica de un FPGA, la cual se encuentra formada por tres elementos básicos:

- CLBs (Bloques de Lógica Configurable), proporcionan los elementos funcionales para construir la mayoría de la lógica así como el almacenamiento de datos. Su complejidad puede variar desde un par de transistores hasta un conjunto de memorias de acceso aleatorio denominadas tablas de consulta (LUT, *Look-Up-Tables*).
- IOBs (Bloques de Entrada/Salida), estos recursos lógicos proveen el control de flujo de datos entre lógica interna y las terminales de entrada/salida del dispositivo.
- Recursos de interconexión, las entradas y salidas de los CLB e IOB se interconectan mediante líneas e interruptores programables que se ubican en los canales de cableado entre las filas y columnas de los bloques.
- Los FPGA's son particularmente adecuados para la implementación de algoritmos paralelos. Sin embargo, los algoritmos secuenciales, especialmente aquellos que no exigen gran capacidad de procesamiento, son más fáciles de implementar como un

programa para un microprocesador, o mediante el uso de procesos en lenguaje de descripción de hardware.

2.7. Principales Características de un FPGA

Los FPGA son dispositivos orientados a satisfacer una muy amplia gama de aplicaciones, desde simple lógica combinacional hasta sistemas con microprocesador embebido, transmisión tipo Ethernet, transmisión de datos series a 3.5Gb/s, todo con el mismo dispositivo. Por ello los FPGA tienen características diversas, pero se podría decir que las principales son las siguientes:

- Gran cantidad de terminales de E/S. Desde 100 hasta unos 1400 terminales de E/S
- Buffers de E/S programables: control de sesgo, control de corriente, configuración del estándar de E/S, pull-up y pull-down configurables.
- Gran cantidad de Flips-Flops, los dispositivos más grandes tienen unos 40.000 FFs.
- Gran cantidad de Tablas de Búsqueda (Look-Up Tables), ~100.000
- Bloques de Memoria (BRAM) de doble puerto, puerto simple, de hasta 18Mbits, configurables como RAM, ROM, FIFO y otras configuraciones
- Bloques dedicados de Multiplicación
- Transceptores para transmisión serie de muy alta velocidad, entre 1.5 a 10.0Gb/s
- Procesador en hardware embebido, tal como el ARM9
- Procesadores descriptos en software, HDL, tales como el 8051, ARM3
- Controladores de reloj tipo Delay Lock Loop (DLL) y Phase Lock Loop (PLLs) de hasta 550MHz. De 2 a 8 controladores por dispositivo.
- Control de impedancia programable por cada terminal de E/S
- Interface DDR/DDR2 SDRAM soportando interfaces de hasta 800 Mb/s
- Interfaz con estándares de E/S tipo diferencial tales como LVDS, SSTL diferencial, entre otras.

2.8. Celda Básica de Configuración del FPGA

El elemento básico de un FPGA desde el punto de visto no-lógico, es decir, que no tiene una función digital lógica, es la celda de configuración. Esta celda es la que va determinar la configuración de cada elemento lógico, por ejemplo si un flip-flop se va usar o no, y en caso de usarlo, si se configura como D o T. La celda de configuración también determina la configuración de los elementos de ruteo y de las interconexiones, tal como se verá más adelante. Existen en la actualidad cuatro tipos de celdas de configuración de un FPGA. Una está basada en una celda SRAM, que como su nombre indica se usa una pequeña celda SRAM para mantener la configuración de cada parte configurable del FPGA. Otro tipo de celda involucra una celda llamada anti-fuse (anti-fusible), que consiste en una estructura microscópica la que, a diferencia de un fusible regular, esta normalmente abierta. Cuando se hace circular una cierta cantidad de corriente durante la configuración del dispositivo, cerca de los 5mA, causa una gran potencia de disipación en un área muy pequeña, lo que provoca el derretimiento de un aislante dieléctrico entre dos electrodos formando una unión permanente muy fina. El tercer tipo de celda de configuración se basa en celdas tipo Flash, que a diferencia de las SRAM, permiten mantener la configuración aún después de desconectada la alimentación del dispositivo. Hay también FPGA que tienen en el mismo dispositivo celdas Flash y celdas SRAM. Las celdas Flash se usan para mantener los datos de configuración del FPGA y las SRAM para la configuración lógica del FPGA.

La celda de configuración tipo SRAM, que como su nombre lo indica, es una pequeña celda SRAM que se usa para mantener la configuración de cada parte configurable del FPGA. La gran ventaja de los FPGA basados en celdas SRAM es que utilizan un proceso de fabricación estándar. La ‘fabrica’ de FPGA-SRAM tiene un procedimiento de fabricación muy conocido, que es el usado en la fabricación de memorias SRAM, lo que debido a la enorme cantidad de memorias SRAM que se producen para el mercado digital, permite lograr costos de producción muy bajos, muy alta performance y trabajar con un proceso de fabricación muy amortizado y de gran rendimiento (yield). Como es sabido las celdas de memoria tipo SRAM pueden ser reprogramadas un sinnúmero de veces, del mismo modo un FPGA basado en celdas de

configuración SRAM puede ser reprogramado un infinito número de veces, aun cuando el FPGA ya esté montado y soldado en un circuito impreso (PCB). Esta reprogramación en PCB se denomina Programable En Circuito (In-Circuit Programmable, ISP). Esta tecnología SRAM es muy útil también para llevar a cabo una actualización rápida del sistema digital dentro del FPGA. Por ejemplo, algunos sintonizadores de televisión satelital usan FPGA en su sistema para sus diversas funciones. Cuando por alguno motivo se necesita actualizar el sistema dentro del FPGA, se envía por satélite un nuevo archivo de configuración. La lógica del sintonizador, que interpreta el comando de actualización, procede a actualizar el archivo de configuración del FPGA (auto-actualización), evitando de este modo tener que cambiar el sintonizador. La gran desventaja de las celdas SRAM es que son volátiles, lo que significa que aún un simple pulso (glitch) en la tensión de alimentación borraría la configuración del FPGA, quedando prácticamente sin ninguna funcionalidad hasta que se lo configure de nuevo. Otra desventaja es que, debido a que la selección del camino de conexión entre los diferentes bloques lógicos (llamado ruteo), se basa en celdas SRAM, se provocan grandes retardos de ruteo, lo que es un problema en diseños que requieren un rendimiento muy alto. Una desventaja más de estos dispositivos es que, en el producto final, una vez que el FPGA está en el PCB y listo para ser comercializado, necesita de una memoria externa pequeña (tipo Flash, serie o paralela) que mantiene el archivo de configuración del FPGA. El FPGA tiene una pequeña MEF que, cuando se le da tensión de alimentación, le indica al FPGA que tiene que ir a buscar la configuración del mismo a una memoria externa. Luego lee la información de la memoria y configura las celdas SRAM. Para algunas aplicaciones, como sistemas médicos de emergencia, este tiempo de lectura de la memoria y configuración es muy largo (50 ~ 500ms). Para otras aplicaciones, como interfaces series, este tiempo pasa desapercibido.

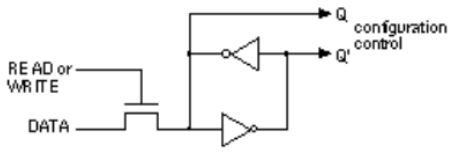


Figura 3. Celda básica de configuración

http://dea.unsj.edu.ar/sisdig2/Field%20Programmable%20Gate%20Arrays_A.pdf

La Figura 3 muestra un ejemplo de la tecnología SRAM de configuración de los FPGA. Esta celda básica está construida por dos inversores cruzados. La salida de la celda de configuración es conectada al transistor de paso, para conectarlo o desconectarlo. La celda se programa usando las líneas Write y Data.

2.9. FPGA basado en celdas Anti-Fuse

Otro tipo de FPGA usa una celda llamada anti-fuse (anti-fusible) como celda básica de configuración. Esta celda consiste en una estructura microscópica que, a diferencia de un fusible regular, está normalmente abierta. Está compuesta de tres capas, las conductoras arriba y abajo, y la aisladora en el medio (Metal-Insulator-Metal, MIM). Para configurar el dispositivo se hace circular una cierta cantidad de corriente (~5mA), lo que causa una gran potencia de disipación en un área muy pequeña, provocando el derretimiento del aislante dieléctrico (tipo Ooxide-Nitride-Oxide, ONO) entre los dos electrodos conductores (SiO2, Dióxido de Silicio) formando una unión permanente muy fina de unos 20nm. La estructura anti-fuse fue creada y se usa habitualmente en ciertas familias de los FPGA de la empresa ACTEL. Esta estructura es conocida como Programmable Logic Interconnect Circuit Element, PLICE.

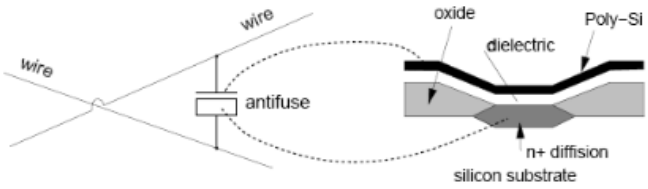


Figura 4. Esquema del anti-fuse de ACTEL
http://dea.unsj.edu.ar/sisdig2/Field%20Programmable%20Gate%20Arrays_A.pdf

La principal ventaja de los FPGA-Anti-Fuse es que no son volátiles, lo que para algunas aplicaciones es sumamente crítico, por ejemplo aplicaciones espaciales y aplicaciones médicas. También, debido a esta tecnología, los retardos de conexión entra los bloques lógicos son muy reducidos, por lo que el rendimiento de estos dispositivos es bastante elevado.

Como desventajas se tienen: primero, que requieren un proceso de fabricación específico, bastante complejo, lo que lleva a que el costo de los mismos sea bastante elevado comparado con los FPGA-SRAM (como mínimo 200 veces más caros). También requieren un programador especial para poder programar el anti-fusible, y la mayor desventaja es que una vez que se han configurado con cierta lógica, están o se puede cambiar, lo que es conocido técnicamente como One Time Programmable, OTP. El hecho de que estos FPGA sean OTP crea un proceso de verificación muy meticuloso de la lógica a ser programada, a fin de no tener que descartar este dispositivo tan caro por errores de diseño. Este proceso de verificación tan largo, implica también incrementar los costos de desarrollo del diseño, al tener que disponer de más horas-ingeniero para tener listo y sin problemas el producto final. Una de las principales aplicaciones de este tipo de FPGA es para sistemas espaciales, ya que estos FPGA son tolerantes a las radiaciones de partículas de alta energía (los bits de configuración no pueden cambiar si son golpeados por una partícula).

2.10. FPGA basados en celdas FLASH

Los FPGA-Flash, tienen como ventaja lo mejor del FPGA-SRAM y del FGPA-Anti-Fuse, son reprogramables y no son volátiles. Sin embargo todavía son caros, ya que usan una tecnología más cara que la SRAM, y las celdas FLASH se usan no solo para guardar la configuración en si del FPGA, sino que también para todo lo que es ruteo, lo que hace que la cantidad de celdas FLASH por FPGA sea un gran número. Los procesos de fabricación de celdas FLASH recién ahora son más comunes. Actualmente en el mercado están apareciendo más opciones de estos dispositivos, sobre todo en los tamaños de FPGA medianos-chicos, pero como la competencia es muy grande y a veces centavos marcan la diferencia, la demanda todavía no es considerable. Otra desventaja para los FPGA FLASH es que el proceso de reconfiguración toma varios segundos. La Figura 5 detalla la estructura de una celda FLASH de la empresa ACTEL, que le llama FLASH Switch. Usa dos transistores que comparten la compuerta flotante, la que almacena la información de configuración. Uno es el transistor de sensado, el cual sólo se usa para escribir y verificar la tensión de compuerta flotante. El otro es el transistor de conexión

(switching). Esta celda puede ser usada en el FPGA para conectar/separar rutas de conexiones o para configurar la lógica.

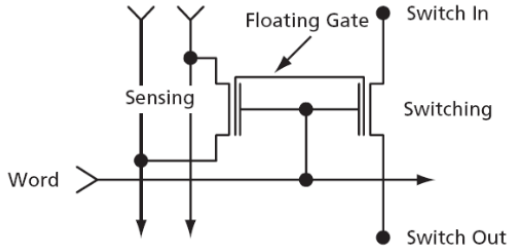


Figura 5. ACTEL flash switch
http://dea.unsj.edu.ar/sisdig2/Field%20Programmable%20Gate%20Arrays_A.pdf

2.11. FPGA basados en celdas Flash y SRAM

Hay algunos FPGA que tienen celdas Flash y SRAM en el mismo dispositivo. Las celdas Flash se utilizan para guardar los datos de configuración del FPGA, mientras que las celdas SRAM para la configuración de la lógica del FPGA. Cuando se da tensión de alimentación, las celdas SRAM se configuran en forma casi instantánea desde la Flash, resultando una configuración del FPGA en menos de 1ms, a diferencia de un FPGA-SRAM cuyo tiempo de configuración típicamente va de los 50 hasta los 500 ms (dependiendo del tamaño del dispositivo). Esta disponibilidad casi instantánea del FPGA lo hace muy útil para aplicaciones críticas en tiempo. Estos FPGA también permiten configurar solo las celdas SRAM, por ejemplo, durante el proceso de construcción del prototipo, sin tener que programar la Flash. Una gran ventaja, y que a veces es decisiva para el diseñador al elegir entre este tipo de FPGA o las FPGA-FLASH, es que, al no tener que acceder a un chip de memoria Flash externo, no hay una conexión física entre el FPGA y la memoria Flash que permita que la configuración del FPGA pueda ser expuesta, y de este modo vulnerada, para una posterior re-ingeniería (o ingeniería inversa) sobre el producto final. Se han descubierto muchos casos de productos copiados a través de la lectura de los datos de configuración (bitstream) disponibles en las rutas del circuito

impreso que hay entre el FPGA y la memoria FLASH (éste problema también está presente en las FPGAs-SRAM).

Por supuesto que estos dispositivos son un poco más caros que los FPGA-SRAM, pero más baratos que los FPGA-FLASH, ya que en este caso las celdas FLASH solo se usan para guardar la configuración, por lo que la cantidad de estas celdas es mucho menor que en el caso de FPGA-FLASH.

2.12. Transición de FPGA a FPGA-ASIC

Finalmente cabe mencionar que empresas como Xilinx, y Altera ofrecen una conversión del FPGA que ya está listo para producción comercial a un dispositivo llamado FPGA-ASIC. Altera le llama HardCopy ASIC, mientras Xilinx le llama EasyPath. Para grandes volúmenes de producción (mayores a 500.000 unidades por año) ésta es una idea atractiva que baja mucho los costos e incrementa la seguridad del diseño (no external bitstream). La arquitectura de este FPGA-ASIC es similar a la arquitectura del FPGA, pero las celdas básicas programables, tanto las de ruteo como las de lógica, ya no son más celdas SRAM o FLASH, ni son programables, son conexiones fijas como en un ASIC. Pero a diferencia de un ASIC, la lógica del sistema que se ha diseñado ya ha sido verificada en hardware, en el FPGA, por lo que el riesgo de tener que re-hacer al ASIC es casi nulo. La parte crítica de esta conversión está referida a los retardos lógicos y de ruteo (delays). Durante la fase de prueba que se lleva a cabo en el FPGA se trabaja con las celdas SRAM, que como ya se sabe agregan bastante retardo, pero al pasar al FPGA-ASIC no hay celdas SRAM que retarden, por lo que el rendimiento del dispositivo cambia, aumentando mucho. Hay nuevas herramientas de software que hacen los cálculos de tiempo de una manera más realista, haciendo que el resultado final sea más previsible en lo que respecta a retardos de tiempo.

Otra ventaja que hace atractiva esta conversión es la reducción de potencia. FPGAs son dispositivos que en general consumen mucha potencia, por lo que para aplicaciones que se alimentan con baterías no son una buena alternativa. Con la conversión a FPGA-ASIC el

consumo de potencia se reduce considerablemente. Sin embargo, se debe mencionar que últimamente los fabricantes están ofreciendo FPGAs de muy bajo consumo, tal como la familia IGLOO de Actel, cuyo mercado es el de los equipos portátiles (Sisterna , 2003) [9].

2.13. Bloque Lógico Configurable (CLB)

Conocidos comúnmente como Bloque Lógico Configurable, (Configurable Logic Block, CLB), estos bloques son la parte lógica mayoritaria dentro de un FPGA. Su estructura macroscópica se detalla en la Figura 6, donde se puede ver que un CLB está constituido por 4 rebanadas (Slices), SLICEM S0, SLICEM S1, SLICEL S2 y SLICEL S3, rutas de conexión para el acarreo matemático, Cin y Cout (carry in, carry out), conexiones a la matriz de conexiones (Switch Matrix) que proveen acceso a las rutas de conexiones generales y conexiones locales entre las rebanadas del mismo CLB y CLBs vecinos.

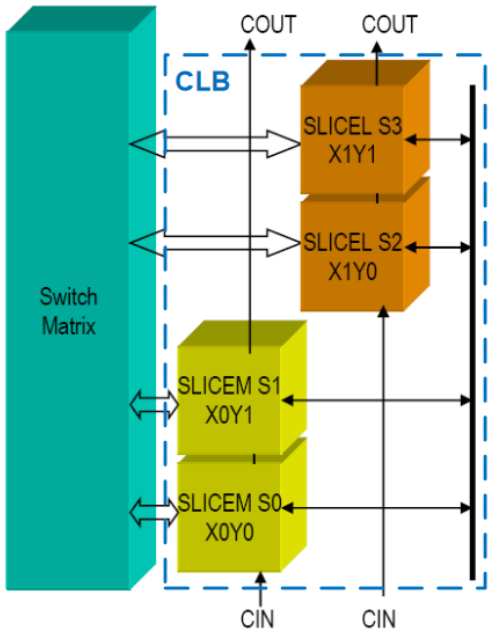


Figura 6. Esquema de un CLB y su conexión a la matriz de conexión
http://dea.unsj.edu.ar/sisdig2/Field%20Programmable%20Gate%20Arrays_A.pdf

Tal como se aprecia en Figura 6, los SLICES se individualizan en coordenadas, desde el más abajo a la izquierda como SLICE X0Y0, hasta el superior derecho identificado como SLICE X1Y1. Del mismo modo para el CLB. Cada CLB se identifica con coordinadas XY dentro del arreglo matricial. Donde XY igual a 00 identifica el primer CLB en la esquina inferior izquierda de la matriz de CLBs, la coordenada del CLB suprior derecho dependerá del tamaño de la matriz, es decir del tamaño del FPGA.

2.14. Rebanadas (SLICE) de un CLB

Hay dos tipos de rebanadas (SLICEs) dentro de un CLB:

- SLICEM: además de las funciones lógicas que se pueden implementar en el SLICE, ofrece opciones para implementar pequeñas memorias.
- SLICEL: solo puede implementar funciones lógicas.

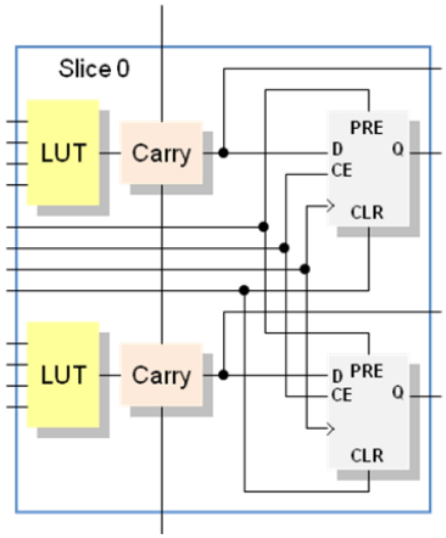


Figura 7. Vista simplificada de un SLICEL

http://dea.unsj.edu.ar/sisdig2/Field%20Programmable%20Gate%20Arrays_A.pdf

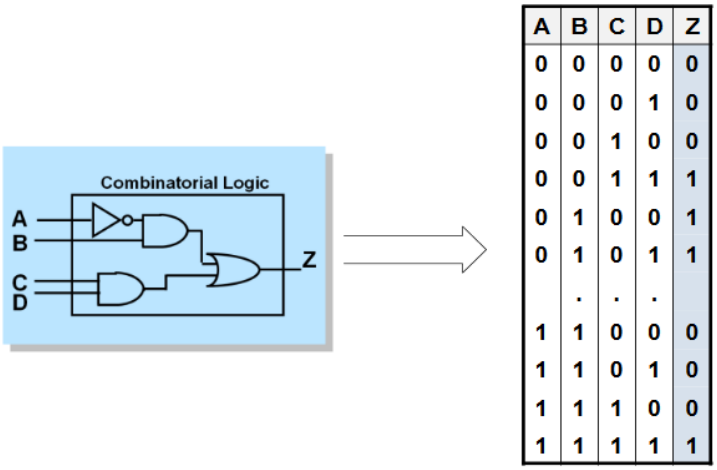
La Figura 7 muestra una vista simplificada de un SLICEL, en la que se destaca lo siguiente:

- Dos tablas de búsquedas (Look-Up Tables)
- Dos flip-flops
- Cuatro salidas, dos combinacionales y dos con registros

- Tiene entradas de control para los flip-flops
- Entradas para las LUTs
- Entrada y salida para la cadena de acarreo (Carry Chain)

2.15. Tablas de Búsqueda -Look-up Tables (LUTs)

En un FPGA toda la lógica combinacional se implementa utilizando tablas de búsqueda, LUT, es decir la función lógica se almacena en una tabla de verdad de 16x1 (para las LUTs de 4 entradas). En cierta literatura a las LUTs también se les llama "Generadores de Funciones" (Function Generators). La Figura 8 detalla la similitud entre una tabla de verdad y una LUT. La columna de valores Z, valores de la función combinacional, son los valores que realmente se almacenan en la LUT de 16x1. Vale recordar que, a menos que por alguna razón se quiera hacerlo manualmente, el almacenamiento de los valoresen las LUTs lo realiza el Software del fabricante del FPGA, siendo el proceso totalmente transparente al diseñador del sistema digital.



A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
	.	.	.	
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Figura 8. Implementación de una función combinacional en una LUT
http://dea.unsj.edu.ar/sisdig2/Field%20Programmable%20Gate%20Arrays_A.pdf

Una característica que a veces es de mucha utilidad, sobre todo para sistemas de muy alta frecuencia, es que el retardo a través de la LUT es constante e independiente de la función implementada.

Para implementar funciones lógicas de más de cuatro entradas en una LUT, se usan multiplexores dedicados que están distribuidos en el SLICE(L/M) y en el CLB para poder implementar cualquier función con un mayor número de entradas. (Chan & Mourad, 1994) [6]

2.16. Interconexiones en los FPGAs

Además de las celdas de lógica programable, los FPGAs tienen celdas de interconexión programables. Estas definen el camino (ruta) a seguir por cada señal interna del FPGA. La tecnológica de configuración y la arquitectura de la celda lógica determinan la estructura y complejidad de la interconexión. La Figura 9 muestra cómo se distribuyen las rutas de interconexión entre los CLBs. A la izquierda se muestra más en detalle cómo es la conexión interna entre las diferentes rutas de conexión, programable a través de transistores.

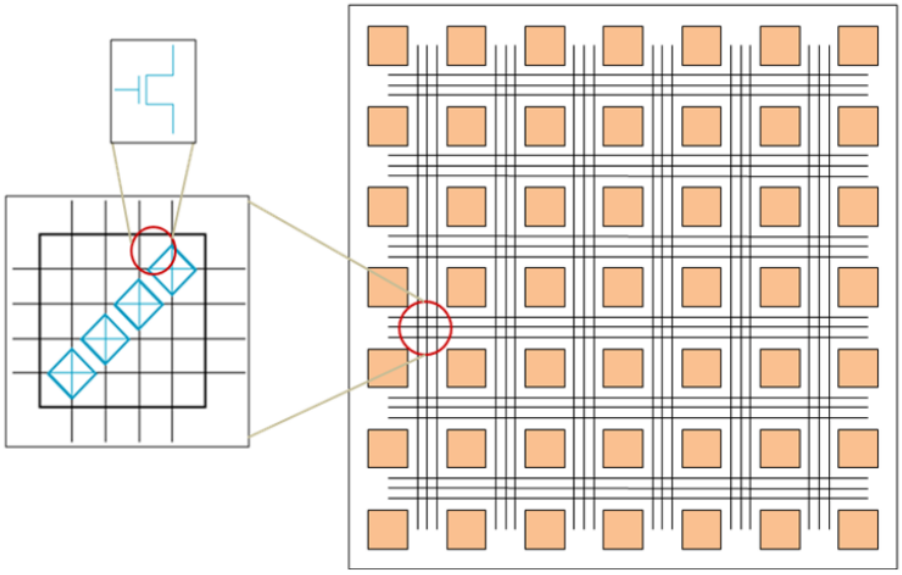


Figura 9. Interconexión entre distintos caminos de ruteo
http://dea.unsj.edu.ar/sisdig2/Field%20Programmable%20Gate%20Arrays_A.pdf

La programabilidad de la interconexión, si bien es una gran ventaja, agrega retardos a la señal que pasa por el transistor. Así, si una señal debe pasar por varios elementos de interconexión la suma total de los retardos puede ser considerable y debe ser tenida en cuenta en diseños de alta frecuencia. Por ello, a fin de no ir sumando demasiados retardos en cada

interconexión, existe lo que se llaman rutas largas. De este modo, si una señal tiene que pasar por más de medio dispositivo puede hacerlo usando estas rutas largas, evitando las interconexiones intermedias. Del mismo modo, existen también las rutas cortas, llamadas conexiones directas, que comunican un bloque lógico con sus bloques vecinos.

Dentro del FPGA también existen lo que se llaman rutas dedicadas, que se usan para las señales que tienen mucha cargabilidad de salida (fan-out) en el sistema que se está diseñando. Las señales que más comúnmente usan las rutas dedicadas son el reloj y la habilitación del reloj (clock enable). Estas rutas tienen buffers especiales que hacen que la señal no se distorsione con la carga. Por ejemplo, en diseños grandes es fácil encontrar una señal de reloj que llegue a 2000 flip-flops.

Hay una relación directa entre la cantidad de interconexiones y el tamaño físico del FPGA. Por ello es que no se implementan físicamente todas las interconexiones posibles, así como, no todos los bloques lógicos están interconectados entre sí. A raíz de esto, para poder conectar la salida de un bloque lógico con la entrada de otro a veces se pasa por diversas interconexiones que conectan diferentes rutas para poder llegar a destino. Por suerte para el diseñador, el software del fabricante del FPGA realiza automáticamente todas las interconexiones. Es una tarea totalmente transparente al diseñador, siempre y cuando se cumplan los requisitos de tiempo estipulados por las especificaciones del sistema implementado. De lo contrario, el diseñador deberá optimizar algunos parámetros de la herramienta de ruteo (llamada Place and Route) para que pueda cumplir con las especificaciones de tiempo requeridas.

2.17. Generación de Reloj y su Distribución

Los FPGA tienen unos bloques de lógica dedicados exclusivamente a funciones de control y generación de señales de reloj. En los FPGAs a estos bloques genéricamente se los llama Digital Clock Managers (DCMs) (Gestores de Reloj Digitales). La cantidad de estos bloques disponibles en un FPGA depende del tamaño del mismo, puede haber desde 2 DCMs en los FPGAs más pequeños hasta 12 DCMs en los FPGA grandes. Los DCMs integran capacidades

avanzadas del reloj, dentro de la red de distribución dedicada del reloj del FPGA. Las principales funciones del DCM se pueden resumir en: Eliminar el sesgo del reloj (clock skew), ya sea dentro del FPGA o con componentes externos. De este modo se mejora el rendimiento del sistema y se eliminan los retardos de ruteo del reloj. Producir corrimiento de fase (Phase shifting) de una señal de reloj, ya sea por una fracción del periodo de reloj o por incrementos fijos. Multiplicar o dividir la frecuencia de entrada del reloj, generando una frecuencia completamente nueva. Acondicionar la señal de entrada del reloj, asegurando un reloj limpio, con un ciclo de trabajo del 50%.Amplificar de nuevo (rebuffer) una señal de reloj, normalmente para eliminar el sesgo (deskew) y convertir la señal de entrada a un estándar diferente, por ejemplo, de LVDS a LVTTL.

CAPÍTULO 3
CONCEPCIÓN DEL SISTEMA

3.1. Tarjeta de Desarrollo (DE2 Development and Education Board)

Este sistema de desarrollo ofrece un amplio conjunto de características que lo hacen adecuado para su uso en una ambiente de laboratorio para la universidad, para una variedad de proyectos de diseño, así como para la Desarrollo de sofisticados sistemas digitales. En este caso, el sistema de desarrollo corresponde a la CPU Cyclone II EP2C35F672C6, además de tutoriales, ejercicios de laboratorio "listos para enseñar" y demostraciones ilustrativas. La figura 10 representa la tarjeta y sus diferentes interfaces.

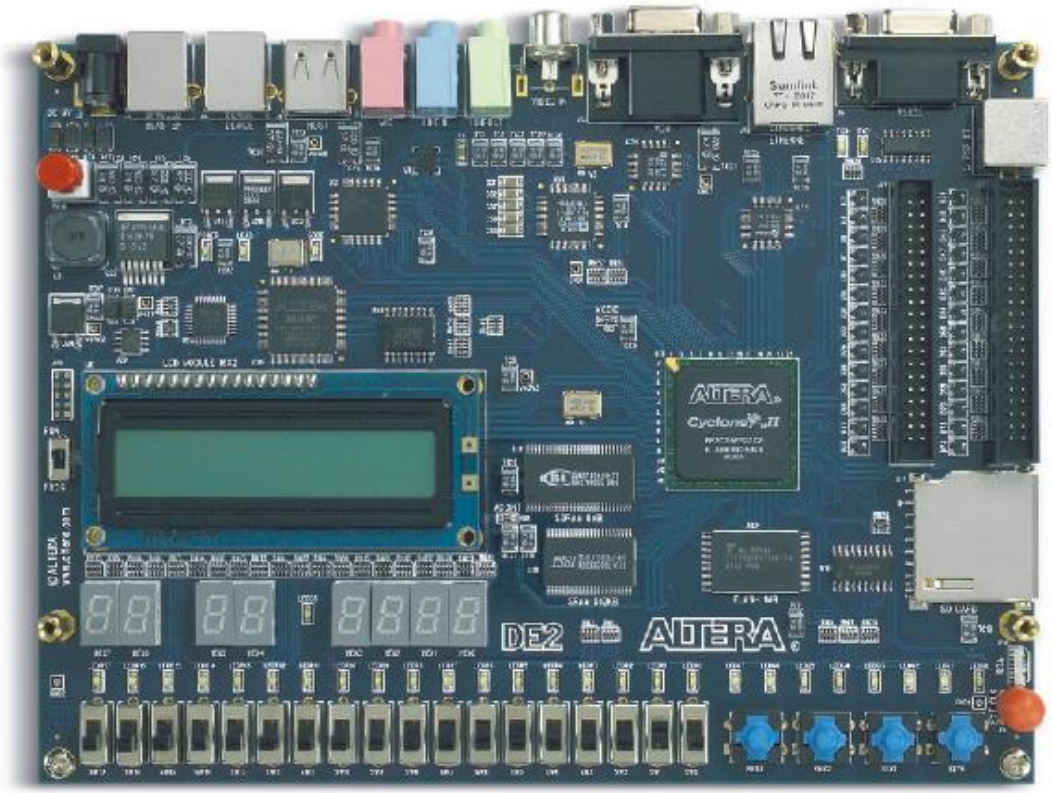


Figura 10. Sistema de desarrollo Cyclone® II 2C35 FPGA

https://images.app.goo.gl/92zn1ZkPbKU4YM9CA

La nomenclatura del dispositivo de la figura 10 para el desarrollo del prototipo en el presente documento se presenta en la figura 11.

```
EP2C35F672C6
 |  |   |  |--> speed grade
 |  |   |-----> Package and pin number
 |  |---------> LEs in 1000
 |------------> Device family
```

Figura 11 Nomenclatura del dispositivo

https://books.google.com.co/books?id=wzYuOF6HFX0C&pg=PA22&lpg=PA22&dq=EP2C35+altera&source=bl&ots=X9EwqDGPkB&sig=ACfU3U3kozU4dpaaV8y30NueC7IN-gmkBg&hl=es-419&sa=X&ved=2ahUKEwilyOOg4sLkAhVtvFkKHXa5DFsQ6AEwDnoECAkQAQ#v=onepage&q=EP2C35%20altera&f=false

En la figura 12 se detalla el hardware que proporciona la placa de desarrollo DE2:

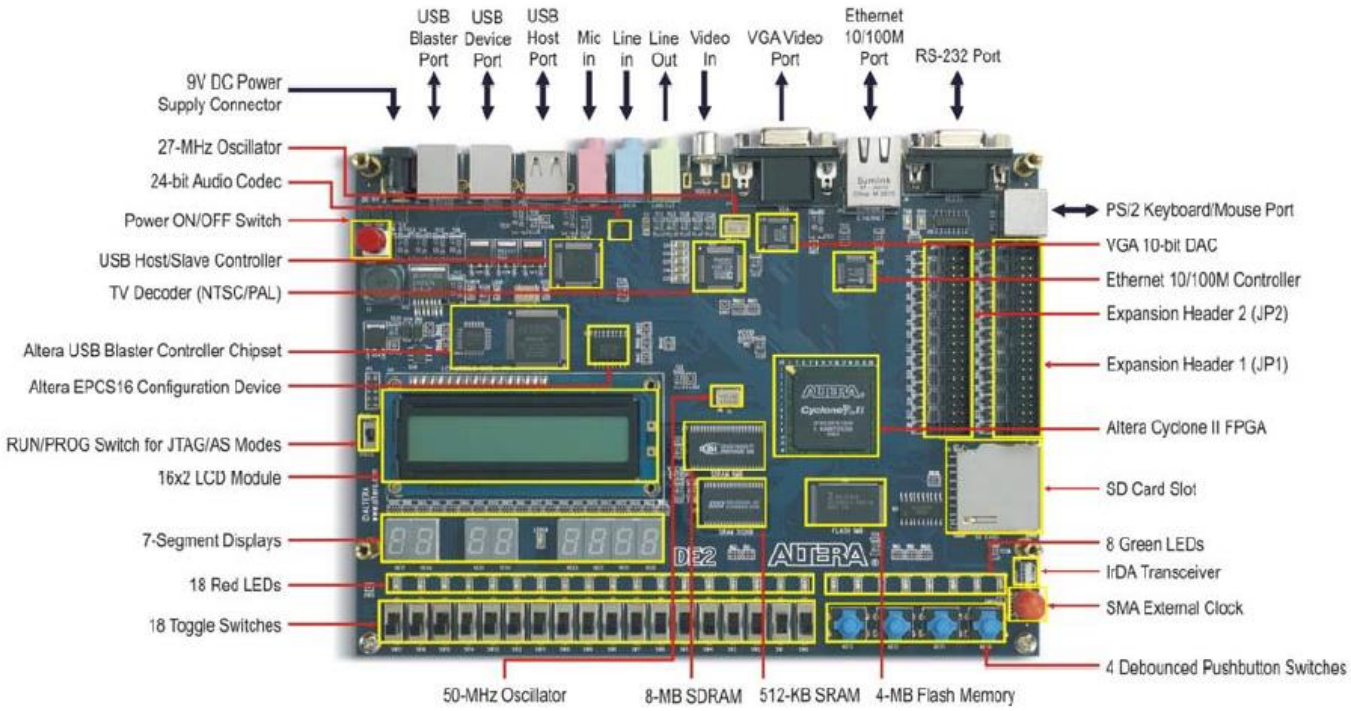


Figura 12. Periféricos de la tarjeta de desarrollo Cyclone II 2C35

https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_cii_starter_board_rm.pdf

El siguiente hardware es proporcionado por la tarjeta de desarrollo:

- Dispositivo Altera Cyclone® II 2C35 FPGA
- Dispositivo de configuración serie Altera - EPCS16
- USB Blaster (a bordo) para programación y control de API de usuario; tanto JTAG como Active Serial(AS) los modos de programación son compatibles.
- 512-Kbyte SRAM
- 8-Mbyte SDRAM
- Manual de usuario DE2
- Memoria Flash de 4 Mbytes (1 Mbyte en algunas tarjetas)
- Socket de tarjeta SD
- 4 pulsadores
- 18 interruptores de palanca
- 18 LEDs rojos de usuario
- 9 LEDs verdes de usuario
- Oscilador de 50 MHz y oscilador de 27 MHz para fuentes de reloj
- CODEC de audio con calidad de CD de 24 bits con tomas de entrada de línea, salida de línea y entrada de micrófono
- DAC VGA (DAC triples de alta velocidad de 10 bits) con conector de salida VGA
- Decodificador de TV (NTSC / PAL) y conector de entrada de TV
- Controlador Ethernet 10/100 con un conector
- Controlador host / esclavo USB con conectores USB tipo A y tipo B
- Transceptor RS-232 y conector de 9 pines.
- PS/2 conector de ratón / teclado
- Transceptor IrDA
- Dos cabezales de expansión de 40 pines con protección de diodo

3.2. Diagrama de bloques de la placa DE2

La figura 13 muestra el diagrama de bloques de la placa DE2. Para proporcionar la máxima flexibilidad para el usuario, todas las conexiones se realizan a través del dispositivo

FPGA Cyclone II. Así, el usuario puede configurar el FPGA para implementar cualquier diseño de sistema (Altera, 2006) [2].

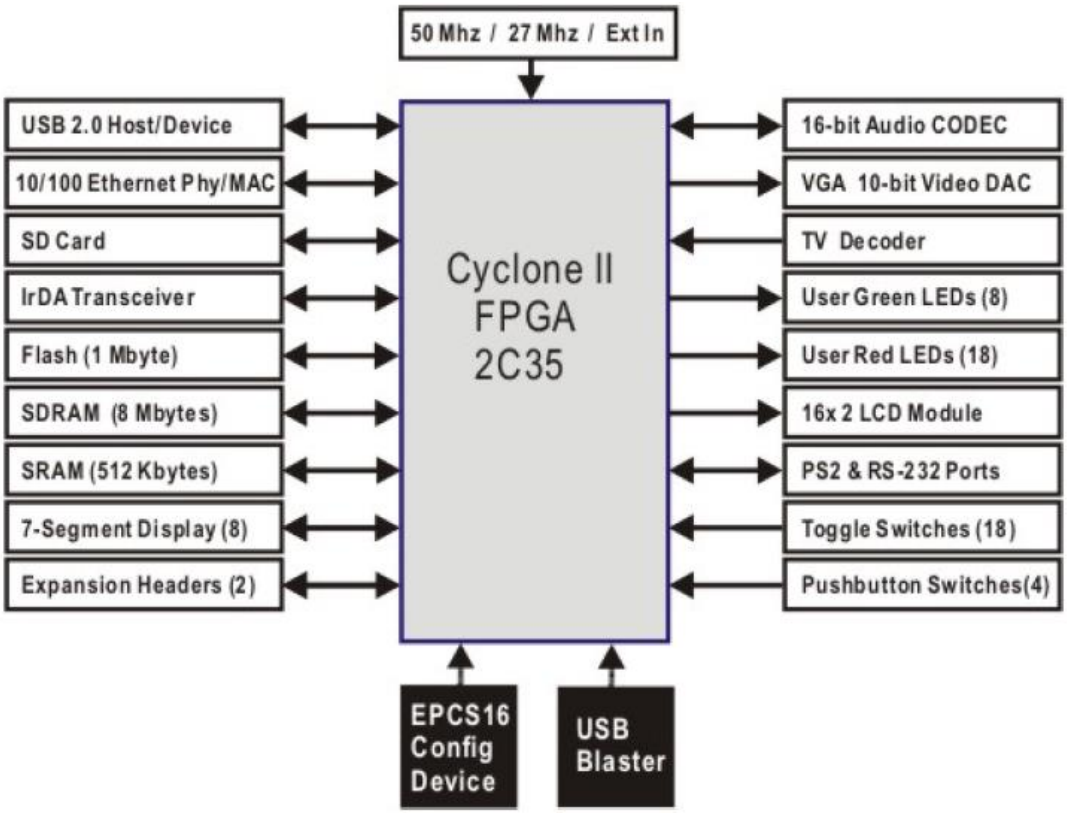


Figura 13. Diagrama de bloques de la tarjeta DE2
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_cii_starter_board_rm.pdf

A continuación, la información más detallada sobre los bloques en la Figura 13:

Cyclone II 2C35 FPGA

- 33,216 LEs
- 105 bloques RAM M4K
- 483,840 bits de RAM totales
- 35 multiplicadores embebidos

- 4 PLLs
- 475 pines de E/S de usuario
- Paquete Fine Line BGA 672-pin
- Dispositivo de configuración serie y circuito USB Blaster
- Dispositivo de configuración serial EPCS16 de Altera
- USB Blaster incorporado para programación y control de API de usuario
- Los modos de programación JTAG y AS son compatibles

SRAM

- 512-Kbyte Static RAM chip de memoria
- Organizado en 256K x 16 bits
- Accesible como memoria para el procesador Nios II y por el panel de control DE2

SDRAM

- 8-Mbyte chip de memoria RAM dinámica síncrona de velocidad de datos única
- Organizado en 1M x 16 bits x 4 bancos
- Accesible como memoria para el procesador Nios II y por el panel de control DE2

Memoria flash

- Memoria Flash NOR de 4 Mbytes (1 Mbyte en algunas tarjetas)
- bus de datos de 8 bits
- Accesible como memoria para el procesador Nios II y por el panel de control DE2

Zócalo de tarjeta SD

- Proporciona el modo SPI para el acceso a la tarjeta SD
- Accesible como memoria para el procesador Nios II con el controlador de tarjeta SD DE2

Interruptores de pulsador

- 4 pulsadores
- Declarado por un circuito de activación de Schmitt.
- Normalmente alto; genera un pulso bajo activo cuando se presiona el interruptor

Interruptores de palanca

- 18 interruptores de palanca para las entradas del usuario
- Un interruptor causa el 0 lógico cuando está en la posición ABAJO (más cercana al borde de la placa DE2) y lógica 1 cuando está en la posición UP

Entradas de reloj

- Oscilador de 50 MHz
- Oscilador de 27 MHz
- Entrada de reloj externo SMA

CODEC de audio

- Wolfson WM8731 CODEC de audio sigma-delta de 24 bits
- Entradas de entrada de nivel de línea, salida de nivel de línea y entrada de micrófono
- Frecuencia de muestreo: 8 a 96 KHz.
- Aplicaciones para reproductores y grabadoras de MP3, PDA, teléfonos inteligentes, grabadoras de voz, entre otros.

Salida VGA

- Utiliza el DAC de video de alta velocidad de 10 bits triple ADV7123 a 240 MHz
- Con conector D-sub de alta densidad de 15 pines
- Admite resolución hasta 1600 x 1200 a una frecuencia de actualización de 100 Hz
- Se puede usar con el Cyclone II FPGA para implementar un codificador de TV de alto rendimiento

Circuito decodificador NTSC / PAL TV

- Utiliza el decodificador de video SDTV multiformato ADV7181B
- Compatible con NTSC- (M, J, 4.43), PAL- (B/D/G/H/I/M/N), SECAM
- Integra tres ADC de 9 bits a 54 MHz
- Timer desde una sola entrada de oscilador de 27 MHz
- Admite entrada de jack RCA de video compuesto (CVBS).
- Admite formatos de salida digital (8 bits / 16 bits): ITU-R BT.656 YCrCb 4: 2: 2 de salida + HS, VS, y FIELD
- Aplicaciones: grabadoras de DVD, televisores LCD, decodificadores, televisores digitales, dispositivos de video portátiles

Controlador Ethernet 10/100

- Direcciones MAC y PHY integradas con una interfaz de procesador general
- Soporta aplicaciones 100Base-T y 10Base-T
- Admite la operación de dúplex completo a 10 Mb/s y 100 Mb/s, con auto-MDIX
- Totalmente compatible con la especificación IEEE 802.3u
- Admite la generación y comprobación de la suma de comprobación de IP/TCP/UDP
- Admite el modo de contrapresión para el control de flujo en modo semidúplex

Controlador USB Host / Slave

- Cumple completamente con la especificación de bus serie universal Rev. 2.0
- Admite la transferencia de datos a velocidad completa y baja
- Soporta dispositivo USB
- Dos puertos USB (uno tipo A para un host y uno tipo B para un dispositivo)
- Proporciona una interfaz paralela de alta velocidad para la mayoría de los procesadores disponibles; soporta Nios II con un controlador Terasic.
- Admite E/S programada (PIO) y acceso directo a memoria (DMA)

Puertos seriales

- Un puerto RS-232
- Un puerto PS/2
- Conector serie DB-9 para el puerto RS-232
- Conector PS/2 para conectar un mouse o teclado PS2 a la placa DE2

Transceptor IrDA

- Contiene un transceptor de infrarrojos de 115.2 kb/s
- 32 mA corriente de accionamiento LED
- Escudo integrado EMI
- IEC825-1 Clase 1 seguro para los ojos
- Entrada de detección de bordes

Dos cabezales de expansión de 40 pines

- 72 pines de E/S Cyclone II, así como 8 líneas a Vcc y tierra, que se llevan a dos conectores de expansión de 40 pines
- El conector de 40 pines está diseñado para recibir un cable tipo ribbon estándar de 40 pines utilizado para los discos duros IDE
- Se proporciona protección de diodo y resistencia

3.3. Recursos Lógicos

El EP2C35 es un miembro de la familia Altera Cyclone II y tiene una densidad lógica equivalente a aproximadamente 35000 elementos lógicos (LE). Hay disponibles 35 multiplicadores adicionales de tamaño 18 × 18 bits (o el doble de este número si se usa un tamaño de 9 × 9 bits).

Tabla 1. Familia de dispositivos Cyclone II de Altera.

Device	Total 4-input LUTs	RAM blocks M4K	PLLs/ clock networks	Emb. mul. 18×18	Max. I/O	Conf. file Mbits
EP2C5	4608	26	2/8	13	89	1.26
EP2C8	8256	36	2/8	18	85	1.98
EP2C20	18 752	52	4/16	26	315	3.89
EP2C35	33 216	105	4/16	35	475	6.85
EP2C50	50 528	129	4/16	86	450	9.96
EP2C70	68 416	250	4/16	150	622	14.31

De la tabla 1 se puede ver que el dispositivo EP2C35 Tiene 33216 elementos lógicos básicos (LEs). Este es también el número máximo de sumadores completos implementables. Cada LE se puede utilizar como una LUT de cuatro entradas, o en modo matemático, como una LUT de tres entradas con un transporte rápido adicional, 16 LEs se combinan siempre en un bloque de matriz lógica (LAB). Por lo tanto, el número de LAB es 33,216 / 16 = 2076. Estos 2076 LABs están organizados en 35 filas y 60 columnas. El dispositivo también incluye tres columnas de bloques de memoria de 4 kbit (llamados bloques de memoria M4K).

El EP2C35 también tiene una columna de multiplicadores de matriz rápidos de 18 × 18 bits, que también puede configurarse como dos multiplicadores de 9 × 9 bits. Como hay 35 filas, el número de multiplicadores es 35 para el tipo de 18 × 18 bits o 70 del tipo de multiplicador de 9 × 9 bits. La Figura 3.3 presenta el plano general del dispositivo.

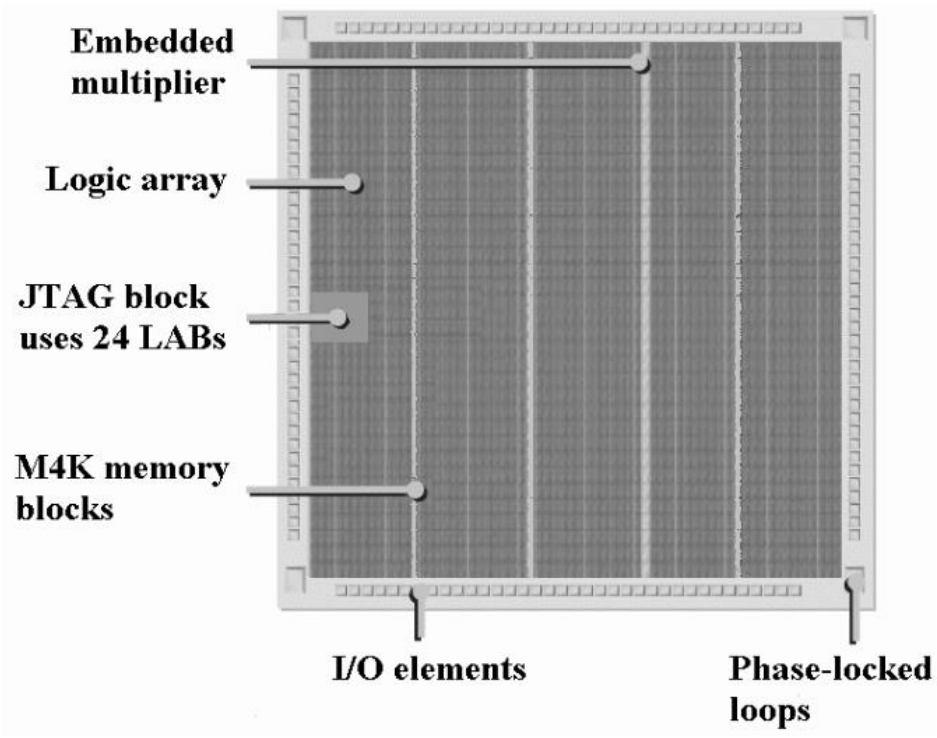


Figura 14. Plano general del dispositivoEP2C35

https://books.google.com.co/books?id=wzYuOF6HFX0C&pg=PA22&lpg=PA22&dq=EP2C35+altera&source=bl&ots=X9EwqDGPkB&sig=ACfU3U3kozU4dpaaV8y30NueC7IN-gmkBg&hl=es-419&sa=X&ved=2ahUKEwilyOOg4sLkAhVtvFkKHXa5DFsQ6AEwDnoECAkQAQ#v=onepage&q=EP2C35%20altera&f=false

3.4. Soft Core

El SoftCore es una descripción del comportamiento de un componente que debe estar sincronizado con las herramientas del proveedor de la FPGA. Normalmente, el bloque se proporciona en un lenguaje de descripción de hardware (HDL) como VHDL o Verilog, que permite que el usuario pueda modificarlo fácilmente, o incluso agregar o eliminar nuevas funciones antes de la síntesis de un dispositivo específico. En el lado negativo, el bloque IP también puede requerir más trabajo para cumplir con los requisitos de tamaño/velocidad/potencia deseados. Muy pocos de los bloques proporcionados por los proveedores de FPGA están disponibles de esta forma, como el microprocesador Nios de Altera o el microprocesador PICOblaze de Xilinx. La protección de IP para el proveedor de FPGA es difícil de lograr, ya que el bloque se proporciona como HDL sintetizable y se

puede usar fácilmente con un conjunto de herramientas/dispositivos de FPGA o un ASIC basado en células.

3.5. Hard Core

El HardCore es una descripción física, provista en cualquiera de una variedad de formatos de diseño físico como EDIF. Los núcleos generalmente se optimizan para un dispositivo específico (familia), cuando se requieren restricciones duras en tiempo real, como por ejemplo una interfaz de bus PCI. Los parámetros del diseño son fijos, como una FFT de 16 bits de 256 puntos, pero una descripción de HDL de comportamiento permite la simulación y la integración en un proyecto más grande. La mayoría de los núcleos IP de terceros de proveedores de FPGA y varios núcleos FFT gratuitos de Xilinx utilizan este tipo de núcleo. Dado que el diseño es fijo, los datos de tiempo y recursos proporcionados son precisos y no dependen de los resultados de la síntesis. Pero el inconveniente es que un cambio de parámetro no es posible, por lo que si la FFT debe tener datos de entrada de 12 o 24 bits, el bloque FFT de 16 bits y 256 puntos no se puede usar.

3.6. Procesador Nios II

El Nios II de Altera es un procesador de software, definido en un lenguaje de descripción de hardware, que se puede implementar en los dispositivos FPGA de Altera usando el programa Quartus II (Sistema CAD).

El procesador Nios II se puede utilizar con una variedad de otros componentes para formar un sistema completo. Estos componentes incluye varios periféricos estándar, pero también es posible definir periféricos personalizados. DE2 de Altera

La tarjera de Desarrollo y Educación contiene varios componentes que pueden integrarse en un sistema Nios II.

Ejemplo de un sistema de este tipo se muestra en la Figura 15.

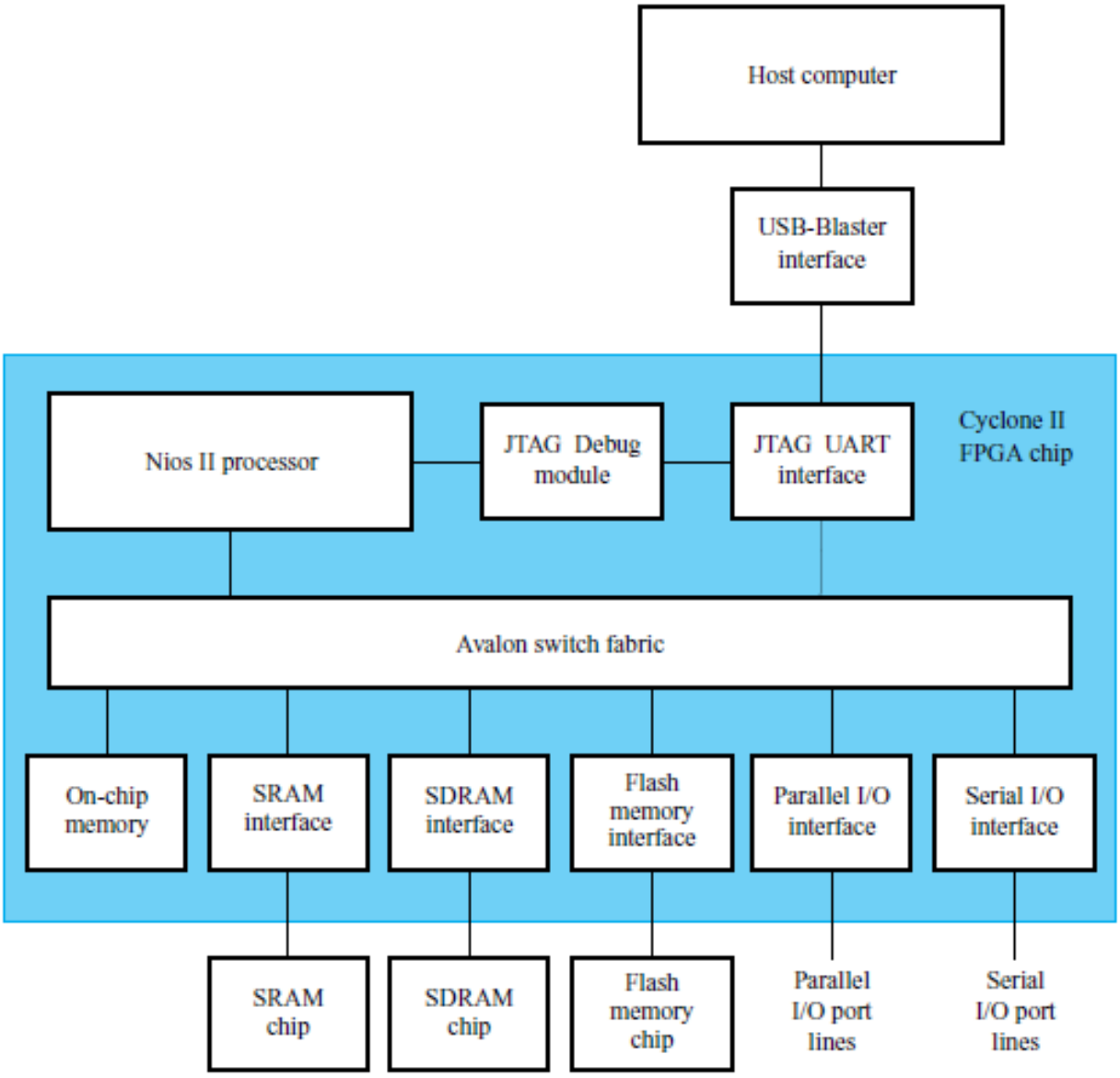


Figura 15. Plano general del dispositivo EP2C35

https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_cii_starter_board_rm.pdf

El procesador Nios II y las interfaces necesarias para conectarse a otros chips en la placa DE2 se implementan en el chip FPGA Cyclone II. Estos componentes están interconectados a través de la

red de interconexión denominada Avalon Switch. Los bloques de memoria en el dispositivo Cyclone II pueden usarse para proporcionar una memoria para el procesador Nios II. Se pueden conectar al procesador directamente o a través de la red Avalon. Las interfaces de entrada y salida están disponibles para proporcionar conexión a los dispositivos de E/S utilizados en el sistema. Una interfaz especial JTAG UART se utiliza para conectarse a la circuitería que proporciona una conexión USB a la computadora. A través del módulo JTAG, el núcleo principal controla el procesador Nios II. Este permite realizar operaciones como descargar programas en la memoria, iniciar y detener la ejecución, configuración de puntos de interrupción del programa y recopilación de datos de seguimiento de ejecución en tiempo real. Dado que todas las partes del sistema Nios II implementado en el chip FPGA se definen mediante descripción de hardware. Con el procesador Nios II sobre la FPGA se realizaron las interfaces con la pantalla y los sensores de cada modelo experimental.

3.6.1. Arquitectura del procesador Nios II

NIOS II es un Softcore desarrollado por la empresa Altera para su implementación en dispositivos SOC y FPGA de la propia marca Altera. Es un procesador de 32 bits de arquitectura RISC.

La figura 16 y 17 muestra la arquitectura en detalle del procesador NIOS II. Esta arquitectura del NIOS II no incluye puerto de acceso a periféricos externos.

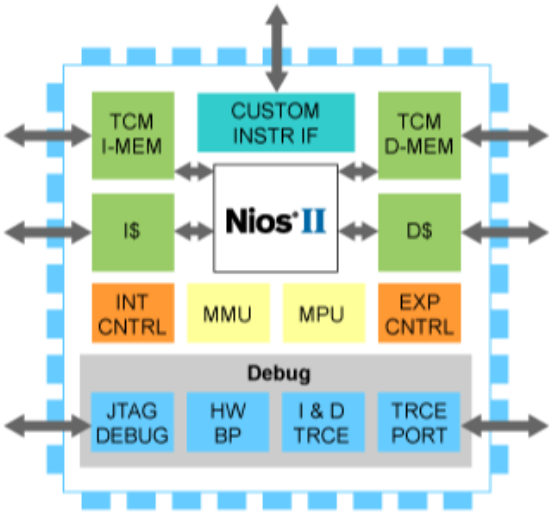


Figura 16. Estructura del SoftCore Nios II

http://www.coffeebrain.org/wiki/images/9/90/Nios-ii-features.png

Una estructura más detallada de la arquitectura de este procesador se muestra en la figura 17.

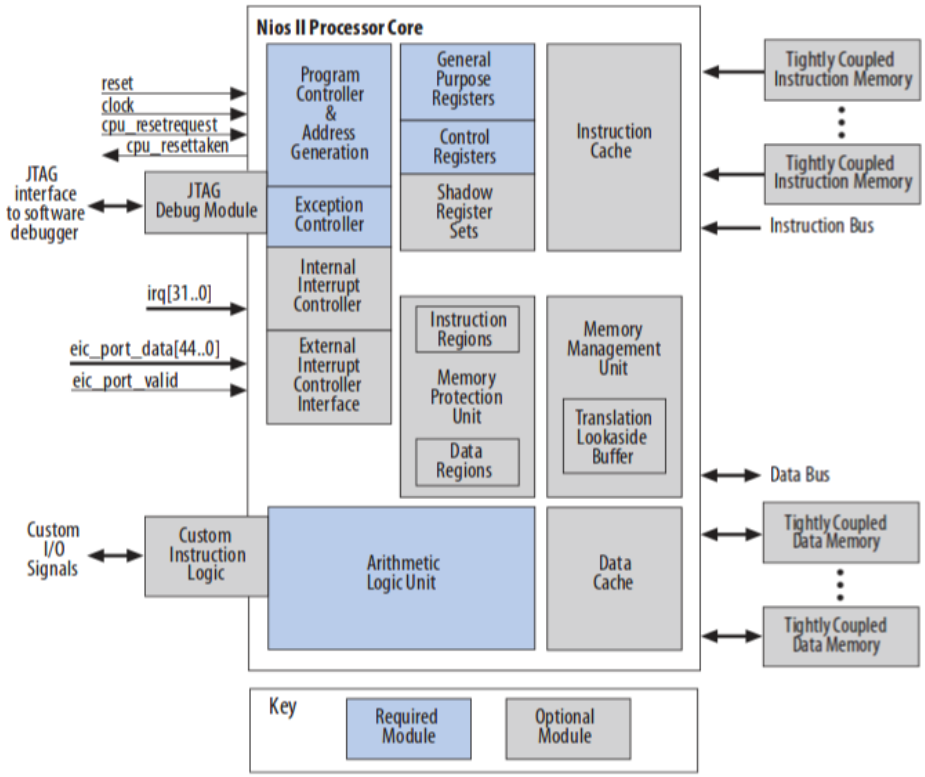


Figura 17. Arquitectura detallada del Nios II SoftCore

http://www.coffeebrain.org/wiki/index.php?title=Archivo:2015-11_NIOSII_2.png

3.7. Hardware usado en el prototipo

3.7.1. Sensor de sonido

Este sensor está compuesto por un Micrófono sensible, referencia KY-037, dos salidas: una llamada "AO", que es una salida analógica, señal de tensión de salida en tiempo real de micrófono y otra llamada "Do", que es una señal digital que se activa cuando la intensidad del sonido a alcanzado un cierto umbral previamente configurado, dicha salida de alta o baja se puede configurar mediante el ajuste del umbral de sensibilidad que puede configurar a través del potenciómetro.

El sensor tiene 3 componentes principales en su placa de circuito. Primero, la unidad de sensor en la parte frontal del módulo que mide el área físicamente y envía una señal analógica a la segunda unidad, el amplificador. El amplificador amplifica la señal, de acuerdo con el valor resistente del potenciómetro, y envía la señal a la salida analógica del módulo.

El tercer componente es un comparador que apaga la salida digital y el LED si la señal cae por debajo de un valor específico. Puede controlar la sensibilidad ajustando el potenciómetro.

Este dispositivo es usado precisamente para detectar el ruido que produce el balín al momento del impacto con la superficie sólida en el modelo de Caída Libre. (CDMX, 2019) [12]

3.7.1.1. *Especificaciones Técnicas*

- Orificios de 3 mm para fijación con tornillos
- Voltaje de funcionamiento: 5 V
- Tiene una salida analógica

- Permite ajustar un nivel de umbral de salida
- Usa el Micrófono Gao Gan, de alta sensibilidad.
- Tiene un indicador LED de encendido
- Tiene un LED que indica la salida del comparador
- interruptor digital salida (0 / 1)
- Dimensiones: 35 x 15 x 14 mm
- Peso: 4 g

3.7.1.2. *Apariencia física*

El aspecto físico del sensor de sonido se muestra en las figura 18a y 18b

a) b)

Figura 18. Sensor de sonido

https://images.app.goo.gl/ceJkK5QztL1kg2Zj9

Los pines de conexión del sensor son:

- En el centro, la conexión a 5V y a GND.
- D0 es una salida digital que actúa a modo de comparador. Si el sonido captado por el micrófono supera un determinado nivel se pone a HIGH.
- A0 es una salida analógica que nos da un valor entre 0 y 1023 en función del volumen del sonido.

Además, dos LEDs, uno que indica si hay alimentación en el sensor y otro que se ilumina si D0 está a HIGH. El ajuste de sensibilidad del micrófono se hace mediante un potenciómetro (trimmer). (PROMETEC, 2019) [20]

3.7.2. Sensor fotoeléctrico

El sensor de CC de la serie SE61 DC fue diseñado para proporcionar una detección confiable, principalmente en aplicaciones OEM directas. Las características del sensor son totalmente de estado sólido y encapsuladas con resina epóxica (polímero termoestable) para una vida ilimitada. El sensor es muy compacto, pero lo suficientemente resistente como para soportar entornos de detección difíciles. Debido a su tamaño pequeño, diseño de ángulo recto y forma única, este par de sensores es ideal para el montaje contra los rieles laterales de los transportadores para uso en control de flujo y detección de producto y es una excelente opción para uso en puestos de pago de supermercados. Es ideal para aplicaciones donde la reflectividad y el perfil del objetivo son suficientes para devolver un gran porcentaje de la luz emitida al sensor. (BANNER, 2019) [11]

3.7.2.1. *Especificaciones técnicas*

- Tecnología Óptica: Infrarrojo
- Tipo de Salida: Bipolar NPN/PNP
- Temperatura de Funcionamiento Mínima: 0 °C
- Temperatura Máxima de Funcionamiento: +50 °C
- Clasificación: IP IP66
- Tensión DC Máxima: 30V
- Tipo detección: Haz pasante (emisor)
- Corriente Máxima: 20000 µA
- Rango de Detección: Máximo de 1,8 m
- Estilo de Sensor: Bloque

- Tiempo de respuesta: 10 ms (apagado), 10 ms (encendido)
- Material de la Carcasa: Poliéster termoplástico reforzado
- Fuente de la luz: LED
- Conexión Eléctrica: Montaje aéreo

La clasificación IP se refiere al grado de protección del sensor ante ciertas condiciones externas como polvo, agua, fuego o golpes. El Grado de protección IP Hace referencia al estándar internacional IEC 60529 Degrees of Protection utilizado con mucha frecuencia en los datos técnicos de equipamiento eléctrico o electrónico, en general de uso industrial como sensores, medidores, controladores, cámaras de seguridad, entre otros.

Este estándar ha sido desarrollado para calificar de una manera alfanumérica a equipamientos en función del nivel de protección que sus materiales contenedores le proporcionan contra la entrada de materiales extraños. Mediante la asignación de diferentes códigos numéricos, el grado de protección del equipamiento puede ser identificado de manera rápida y con facilidad.

Las letras "IP" identifican al estándar (del inglés: International Protection, Protección Internacional). El valor "6" en el primer dígito numérico describe el nivel de protección ante polvo, en este caso: "El polvo no debe entrar bajo ninguna circunstancia".

El valor "6" en el segundo dígito numérico describe el nivel de protección frente a líquidos (normalmente agua), en nuestro ejemplo: No debe entrar el agua arrojada a chorros (desde cualquier ángulo) por medio de una boquilla de 12,5 mm de diámetro, a un promedio de 100 litros por minuto y a una presión de 100kN/m² durante no menos de 3 minutos y a una distancia que no sea menor de 3 metros. Como regla general se puede establecer que cuando mayor es el grado de protección IP, más protegido está el equipo.

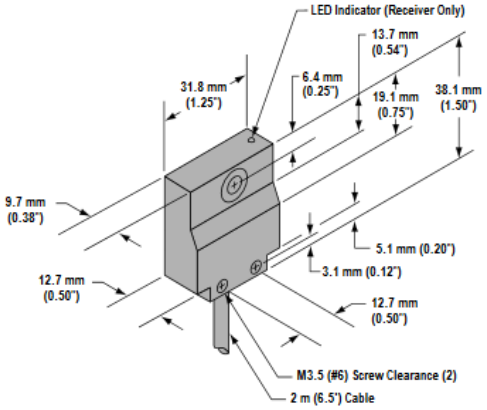
Actualmente la mayoría de los sensores inductivos, capacitivos y fotoeléctricos que se comercializan en el mercado tienen un nivel de protección mínimo de IP66, los cuales los hacen

aptos para soportar la mayoría de los ambientes agresivos que se dan en la industria. (REDATEL, 2018) [22]

3.7.2.2. *Apariencia física y dimensiones*

El aspecto físico del sensor fotoeléctrico así como sus dimensiones se muestran en las figuras 19a y 19b respectivamente (BANNER, 2019) [11].

a) Aspecto físico



b) Dimensiones

Figura 19. Sensor fotoeléctrico [11]

3.7.3. **Sensor de ultrasonido** (Medición de distancia)

El sensor de ultrasonido usado en el instrumento prototipo con referencia US-015, presenta una resolución superior de 1 mm hasta 0,5 mm, con una precisión de alcance; tiene un rango sin contacto de 2 cm ~ 4 m, la tensión de alimentación de 5 V, la corriente de trabajo de 2,2 mA y admite el modo de comunicación GPIO.

3.7.3.1. *Especificaciones técnicas*

- Tensión de funcionamiento: 5 VCC
- Corriente de trabajo: 2,2 mA
- Temperatura de funcionamiento: 0 ~ +70 °C

- Modo de salida: GPIO
- El ángulo de detección: inferior a 15 grados
- Distancia de detección: de 2 cm - 400 cm
- Precisión de detección: 0,1 cm + 1%
- Resolución superior a 1 mm (hasta 0,5mm)

3.7.3.2. *Apariencia física y dimensiones*

El sensor usado para medir distancia en el modelo de la Ley de Hooke se presenta en la figura 20.

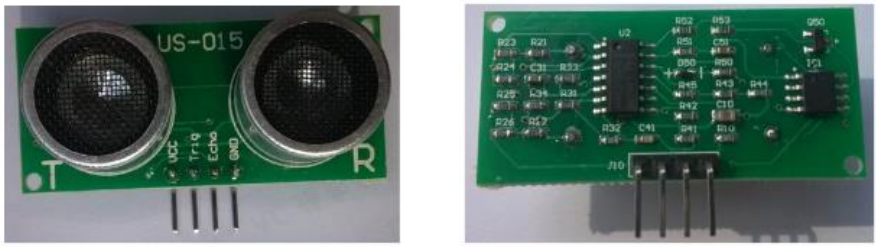


Figura 20. Aspecto físico del sensor de distancia
https://www.addicore.com/Ultrasonic-US-015-Distance-Sensor-Module-p/ad484.htm

Las dimensiones del módulo son: 45mm*20mm*1,2mm. Hay dos orificios mecánicos con un radio de 1 mm en la placa, como se muestra en la Figura 21.

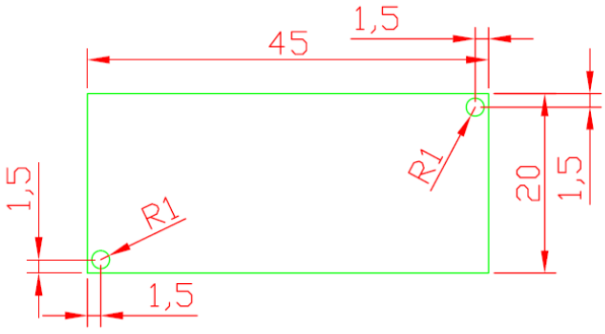


Figura 21. Dimensiones del módulo de distancia

3.7.3.3. *Descripción de la interface*

El módulo tiene una interfaz de 4 pines para fuente de alimentación y comunicación, como se observa en la figura 22.

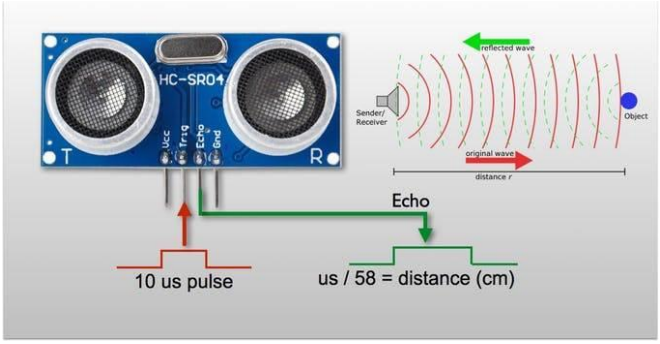


Figura 22. Interfaz del módulo de distancia

https://images.app.goo.gl/wc8pgR5N96j9fToi6

De izquierda a derecha, los pines se definen de la siguiente manera:

1: Alimentación del módulo, 5V DC.

2: **El pin trigger** recibe un pulso de habilitación de parte del sistema, mediante el cual se le indica al módulo que comience a realizar la medición de distancia.

3: **En el pin echo,** el sensor devuelve al sistema un pulso cuyo ancho es proporcional al tiempo que tarda el sonido en viajar del transductor al obstáculo y luego de vuelta al módulo.

4: Referencia o tierra del circuito

Mediante una fórmula puede estimarse entonces la distancia entre el sensor y el obstáculo si se conoce el tiempo de viaje del sonido así como la velocidad de propagación de la onda sonora. La figura 23 muestra los pulsos recibidos y enviados por el sensor.

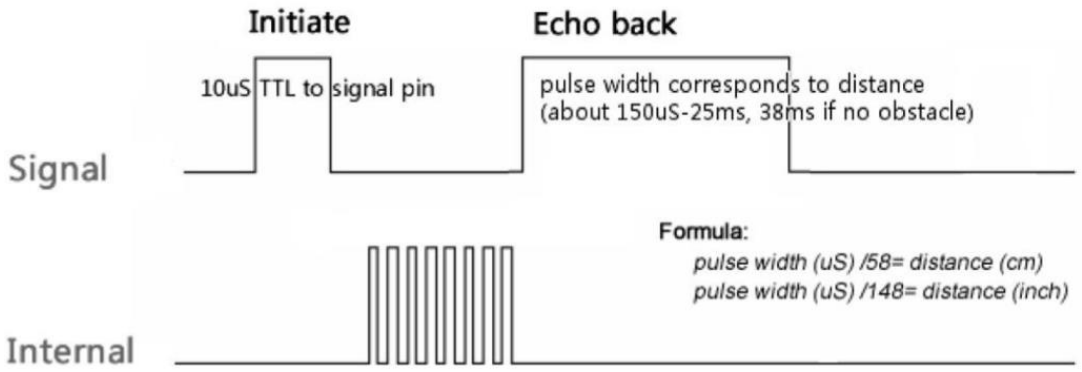


Figura 23. Pulsos del sensor
https://images.app.goo.gl/y1MSTSrunarFAdzh8

Como se puede observar, el sensor genera un pulso en el pin marcado como "echo" cuya duración es proporcional a la distancia medida por el sensor. Para obtener la distancia en centímetros, solamente se divide el tiempo en microsegundos entre 58 o para obtener la distancia en centímetros (o 148 para pulgadas) (Geek Factory, 2018) [14].

3.7.4. Pantalla HMI Nextion

3.7.4.1. *Descripción general de la HMI*

Nextion es una solución HMI (Machine de Human Machine Interface) que combina un procesador integrado y una pantalla táctil de memoria con el software Nextion Editor para el desarrollo de proyectos GUI HMI.

Con el software NEXTION Editor, puede desarrollar rápidamente la GUI de HMI mediante componentes de arrastrar y soltar (gráficos, texto, botones, controles deslizantes, etc.) e instrucciones basadas en texto ASCII para codificar cómo interactúan los componentes en el lado de la pantalla. (NEXTION, 2019) [18]

La pantalla HMI de Nextion se conecta a la MCU periférica a través de la serie TTL (5V, TX, RX, GND) para proporcionar notificaciones de eventos en las que la MCU periférica puede

actuar, la MCU periférica puede actualizar fácilmente el progreso y el estado a la pantalla. Nextion utilizando instrucciones ASCII simples basadas en texto, por esta razón, es una gran solución para monitorear y controlar los procesos, en este caso, la selección del modelo de la Física que se quiere comprobar.

Hay varios módulos de visualización Nextion, con tamaños que van desde 2.4 "a 7". El Nextion tiene un microcontrolador ARM incorporado que controla la pantalla, por ejemplo, se encarga de generar los botones, crear texto, almacenar imágenes o cambiar el fondo. El Nextion se comunica con cualquier microcontrolador mediante comunicación en serie a una velocidad de 9600 baudios. Puede funcionar con cualquier dispositivo que tenga puerto serial. La figura 24 ilustra esta situación. (Random Nerd Tutorials, 2019) [21]

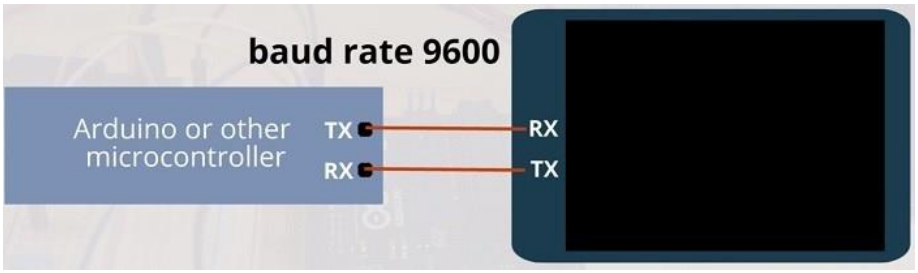


Figura 24. Interfaz de la pantalla Nextion
https://images.app.goo.gl/nnu44y9LvuBzHFT6A

En el capítulo 4 se explicarán en detalle los pasos en el editor Nextion para agregar botones, indicadores, barras de progreso, etiquetas de texto. Tenemos el modelo básico de pantalla Nextion de 2.8 ”, que se muestra en la figura 25.

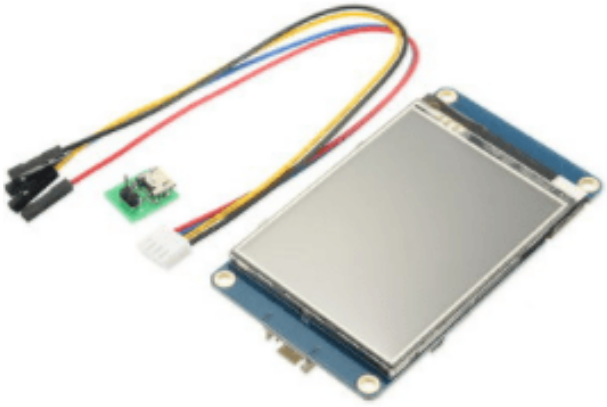


Figura 25. Aspecto físico de la pantalla.

En La figura 26 se muestra los pines de conexión a la tarjeta de control.

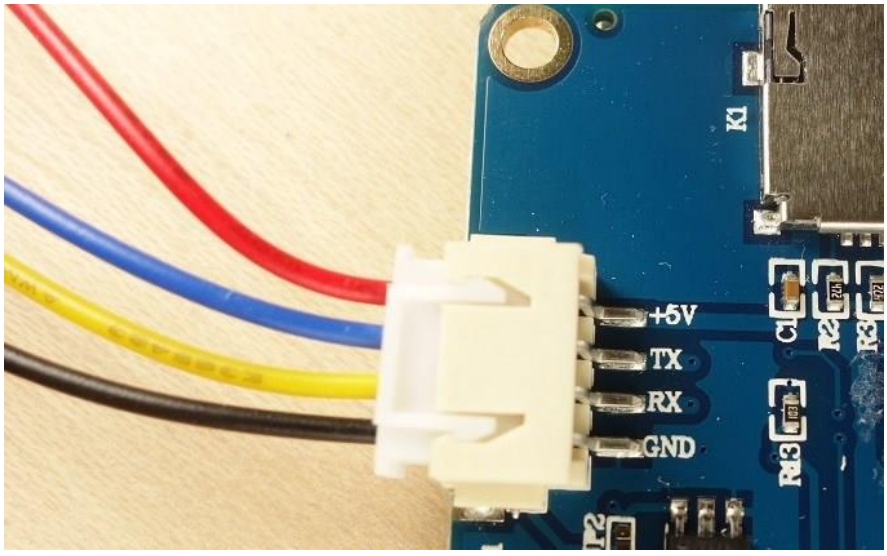


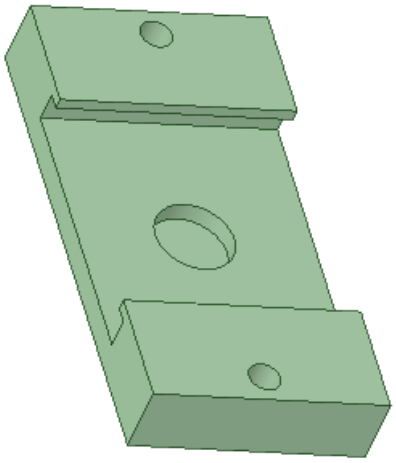
Figura 26. Bornes de conexión a la tarjeta de control

3.8. Diseño de los elementos del instrumento prototipo

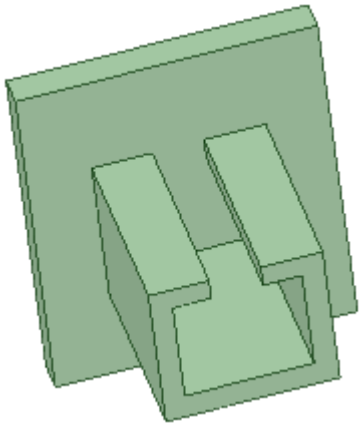
3.8.1. Diseño de módulos de sensores

En la Figura 27 se presentan los diseños iniciales que posteriormente fueron objeto de modificación debido a otras características que mejoran la medición y el cálculo de las variables en cuestión.

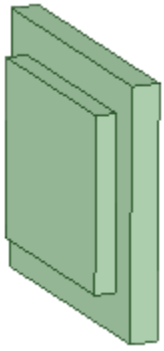
Las partes de los diferentes módulos fueron realizados con el software Design Spark Mechanical v4.0 que es totalmente gratuito, apto para la impresión de modelos 3D e importación de otros tipos de archivos (RS Components Ltd., 2019) [23].



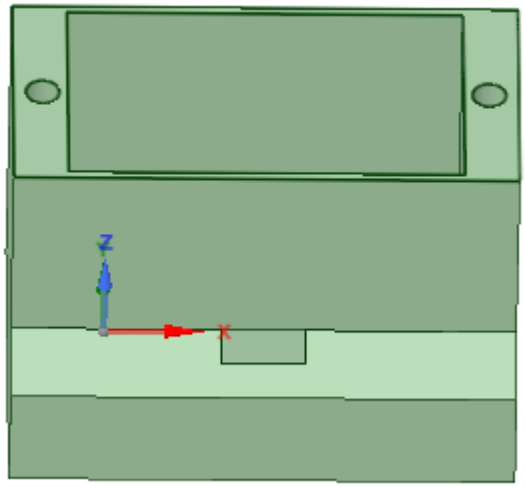
a) Soporte módulos



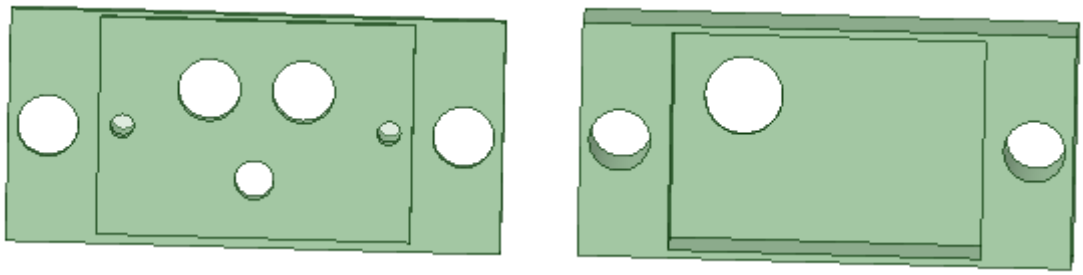
b) Módulo electroimán



c) Tapa módulo electroimán



d) Diseño caja de soporte a verticales



f) Tapa de sensores movibles e) Caja de sensores movibles

Figura 27. Diseño caja y soporte de sensores móviles

3.8.2. Tarjeta de interfaz con la FPGA

La tarjeta se diseñó usando el software simulación Proteus, (ver figura 32), donde se especifican los diferentes conectores para recibir señales de los sensores y enviarlas a la FPGA. Dicha tarjeta consta de los siguientes elementos:

- ATMEGA 328P
- MAX 232
- 4 capacitores de 100µF 16V
- 3 resistencias de 10 kΩ
- 1 capacitor de 100 nF
- 4 conectores de 4 pines

3.8.2.1. *Dispositivo ATMEGA 328P*

Para la interfaz de los sensores y el módulo Nextion se utilizó un procesador ATMEGA328 (Figura 28). Se trata de un microcontrolador basado en AVR RISC de 8 bits de Microchip de alto rendimiento que combina memoria flash ISP de 32KB con capacidades de lectura y escritura, 1KB EEPROM, 2KB SRAM, 23 líneas de E / S de propósito general, 32 registros de trabajo de propósito general, tres temporizadores flexibles / contadores con modos de comparación, interrupciones internas y externas, USART programable en serie, una interfaz serie de 2 hilos orientada a bytes, puerto serie SPI, convertidor A / D de 6 bits de 6 bits (8 canales en

paquetes TQFP y QFN / MLF), temporizador de vigilancia programable con oscilador interno y cinco modos de ahorro de energía seleccionables por software. El dispositivo funciona entre 1.8-5.5 voltios (Microchip, 2015) [17].

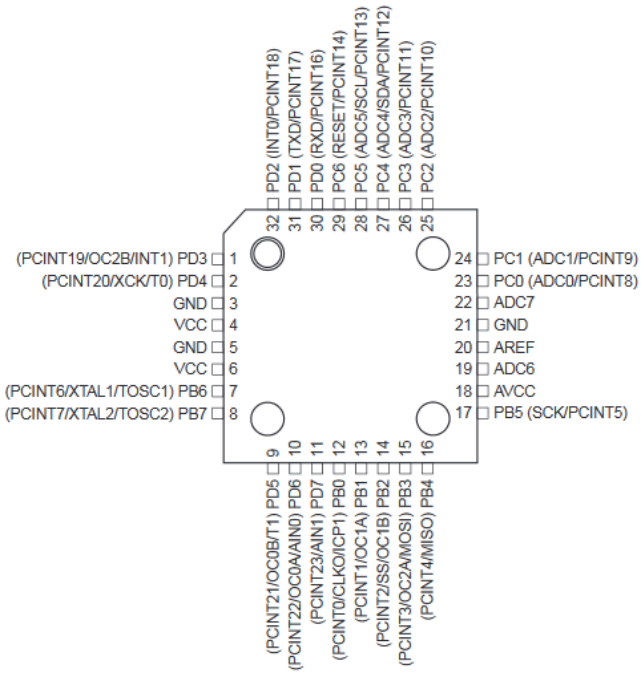


Figura 28. ATMEGA 328P

https://images.app.goo.gl/4u1jJd27764puY1s7

3.8.2.2. *Interfaz MAX232*

El MAX232 como el de la figura 29 es un circuito integrado que convierte los niveles de las líneas de un puerto serie RS232 a niveles TTL y viceversa. Lo interesante es que sólo necesita una alimentación de 5V, ya que genera internamente algunas tensiones que son necesarias para el estándar RS232. Otros integrados que manejan las líneas RS232 requieren dos voltajes, +12V y -12V (Texas Instruments, 2019) [25].

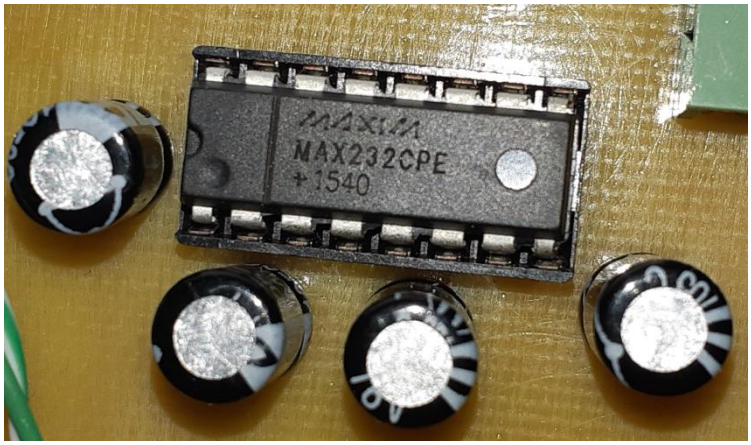


Figura 29. Interfaz serial

El MAX232 soluciona la conexión necesaria para lograr comunicación entre el puerto serie de una PC y cualquier otro circuito con funcionamiento en base a señales de nivel TTL/CMOS. El circuito integrado posee dos conversores de nivel TTL a RS232 y otros dos que, a la inversa, convierten de RS232 a TTL. Estos conversores son suficientes para manejar las cuatro señales más utilizadas del puerto serie del PC, que son TX, RX, RTS y CTS. TX es la señal de transmisión de datos, RX es la de recepción, y RTS y CTS se utilizan para establecer el protocolo para el envío y recepción de los datos. En la figura 30 se aprecia su diagrama de pines y esquema básico de configuración.

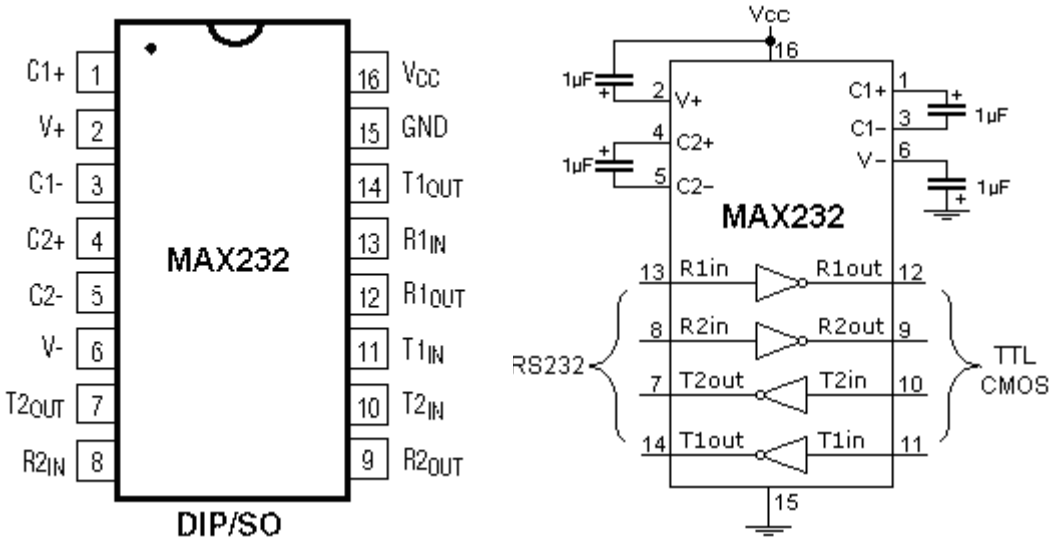


Figura 30. Esquema MAX232

https://images.app.goo.gl/mFVQQQNXKmAWk4yQ9

3.8.2.3. *Optoacoplador de propósito general*

La serie PC817X corresponde a una familia de optoacopladores unido ópticamente a un fototransistor. Se empaqueta en un DIP de 4 pines, en este caso, para transmisión de señales entre circuitos de diferentes voltajes e impedancias. Dicho dispositivo se observa en la figura 31.

Figura 31. Optoacoplador PC817
http://www.ces-eshop.com/dir/image/cache/data/1/Opto%20817-500x500.jpg

El optoacoplador basa su funcionamiento en el acoplamiento óptico de un diodo emisor de infrarrojos (IRED) de Galio-Arsenio, con un Transistor Bipolar de tipo NPN como driver de salida. Este acoplador óptico aísla galvánicamente su entrada de su salida gracias a que internamente, su diodo emisor de luz (LED) excita la base del optotransistor cada vez que se ilumina, garantizando así el aislamiento eléctrico de la parte digital de control (5-12VDC) de la salida de potencia controlada por el optotransistor (COMPIC Electrónica, 2018) [13].

Dentro de las especificaciones para este optoacoplador se encuentran las siguientes:

- La entrada admite inversiones de polaridad.
- Es ideal como relé estático, para el control de cargas en DC. de hasta 35V y 50mA.
- Ofrece un muy alto aislamiento de hasta 5.000V.

- Especialmente diseñado para su empleo como driver de salida DC y como interface para el control de periféricos en sistemas basados en microprocesador.

- Fabricado en formato DIP (Dual In-line Package) standard de plástico de 4 terminales.

3.8.3. Funcionamiento de la Interfaz con la FPGA

Los pines marcados como Audio, Sensor IR, Trigger y Echo son señales provenientes de los sensores como el micrófono, ultrasonido, e infrarrojo que llega a al módulo ATMEGA a través de los pines de datos (D8 a D11) como se observa en la figura 32.

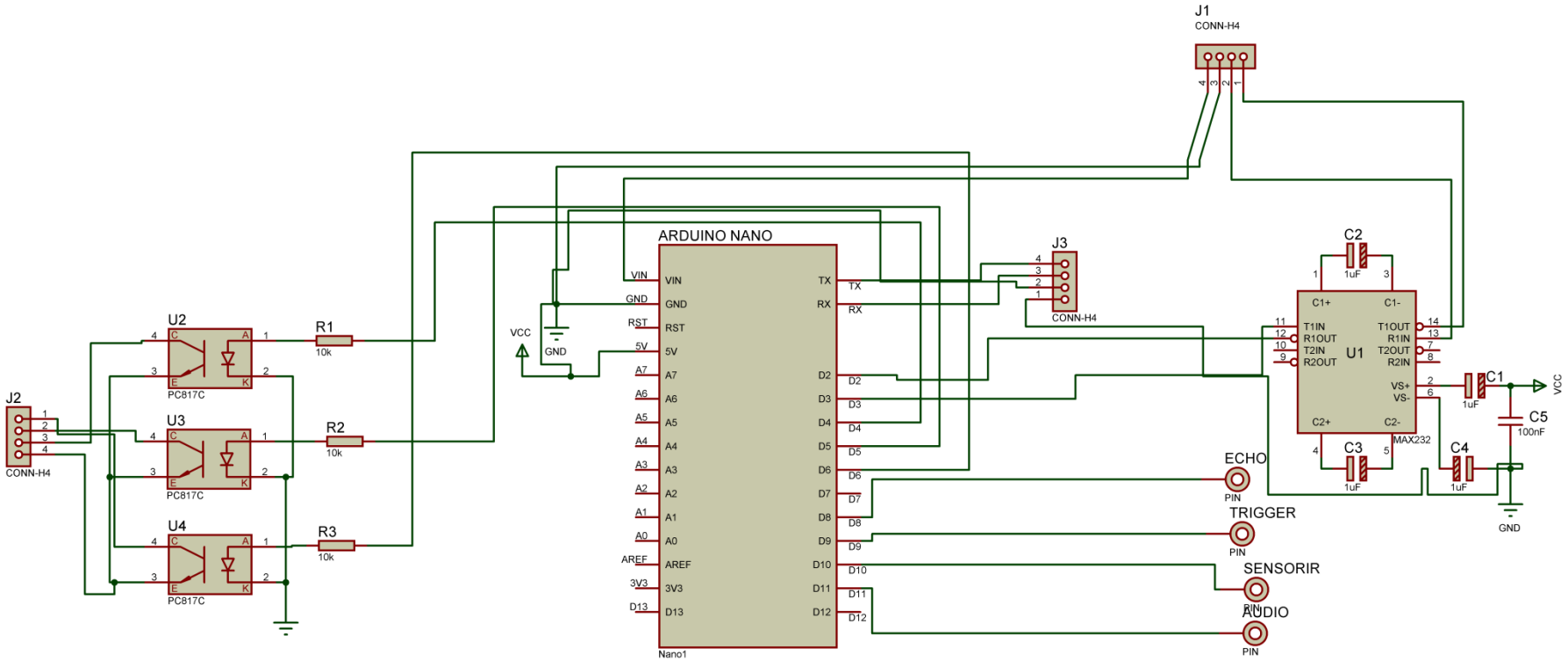


Figura 32. Circuito Interfaz

El esquema mostrado en la figura 32, fue realizado mediante el software de simulación PROTEUS, en cuyos diferentes aplicativos de elaboró la placa de circuito impreso, como se observa en la figura 33, una vista preliminar en la figura 34 y la tarjeta real en la figura 35.

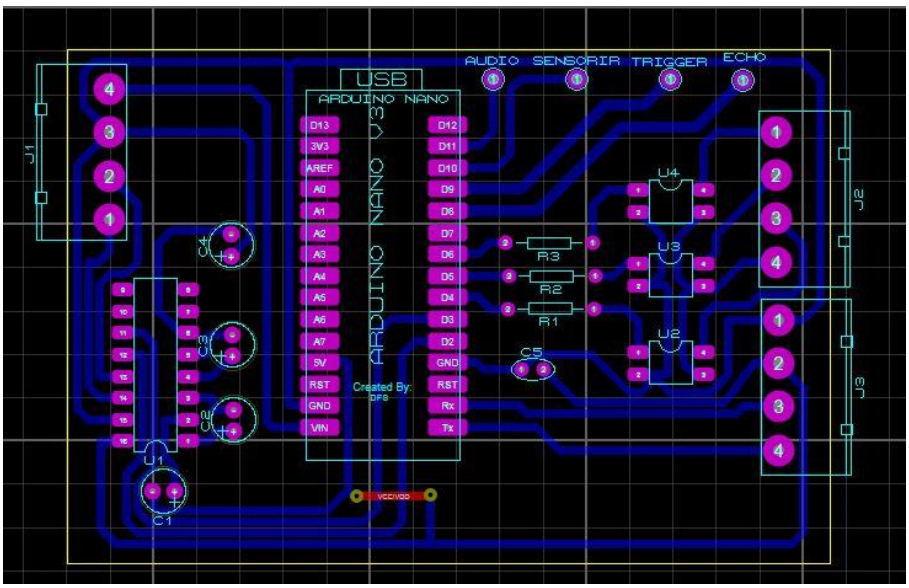


Figura 33. PCB de la tarjeta auxiliar

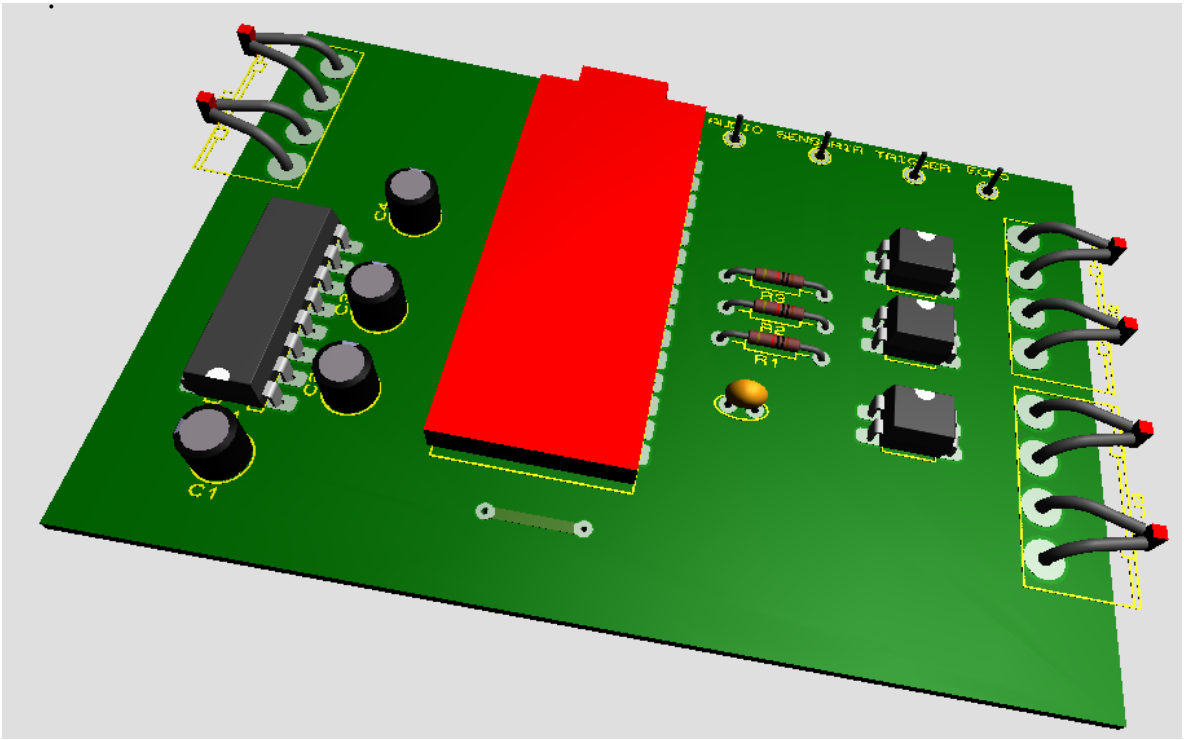


Figura 34. Visualización 3D de la interface

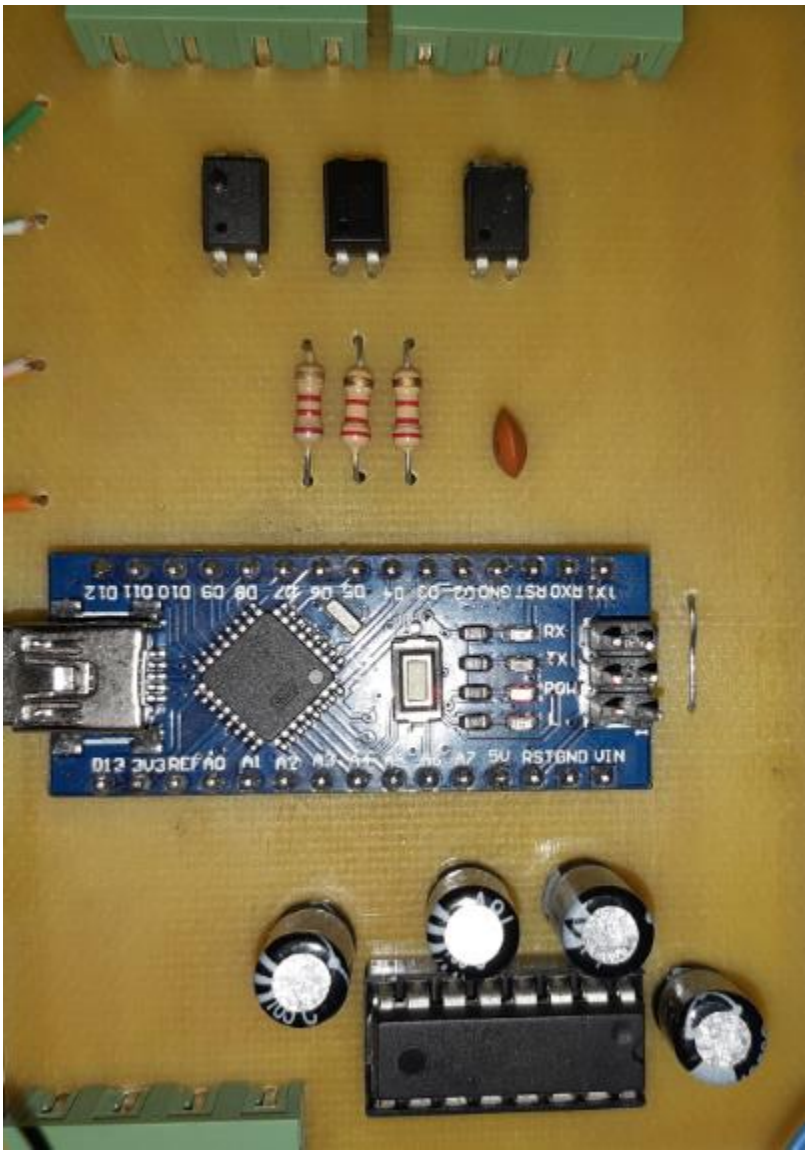


Figura 35. Implementación de la tarjeta auxiliar a la FPGA

Se implementaron las rutinas de programación para el microprocesador de la Nextion en Arduino, las cuales se mostrarán en detalle en el siguiente capítulo.

3.9 Diagrama General del sistema a implementar

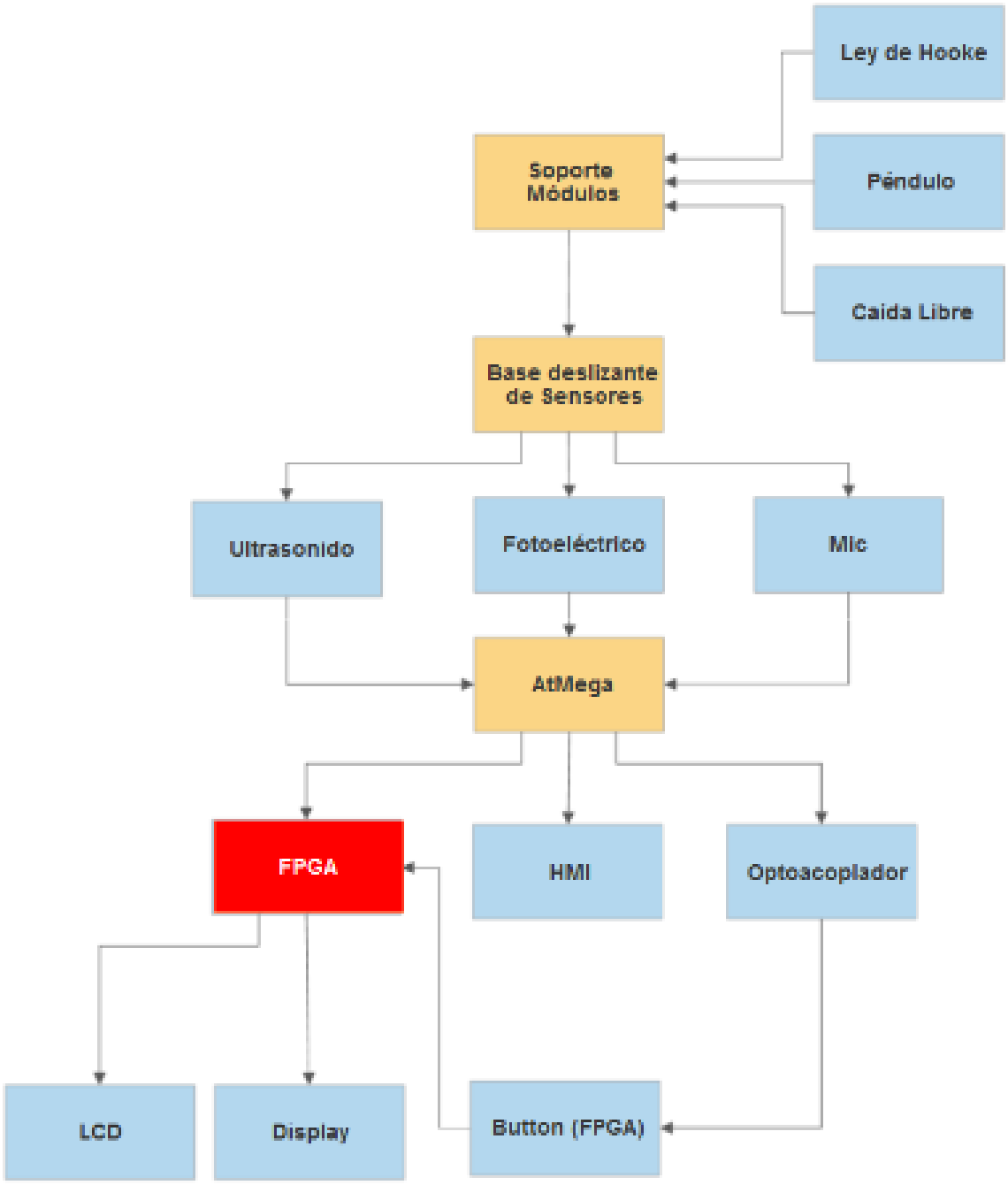


Figura 36. Diagrama de bloques del sistema

CAPÍTULO 4
PROGRAMACIÓN DEL FIRMWARE

4.1. Software Empleado

Para la compilación de los programas usados para configurar los dispositivos se encuentran algunos como Quartus II de Altera, Nextion Editor.

4.1.1. Interfaz Nextion Editor

Este programa presenta la interfaz gráfica mostrada en la figura 36:

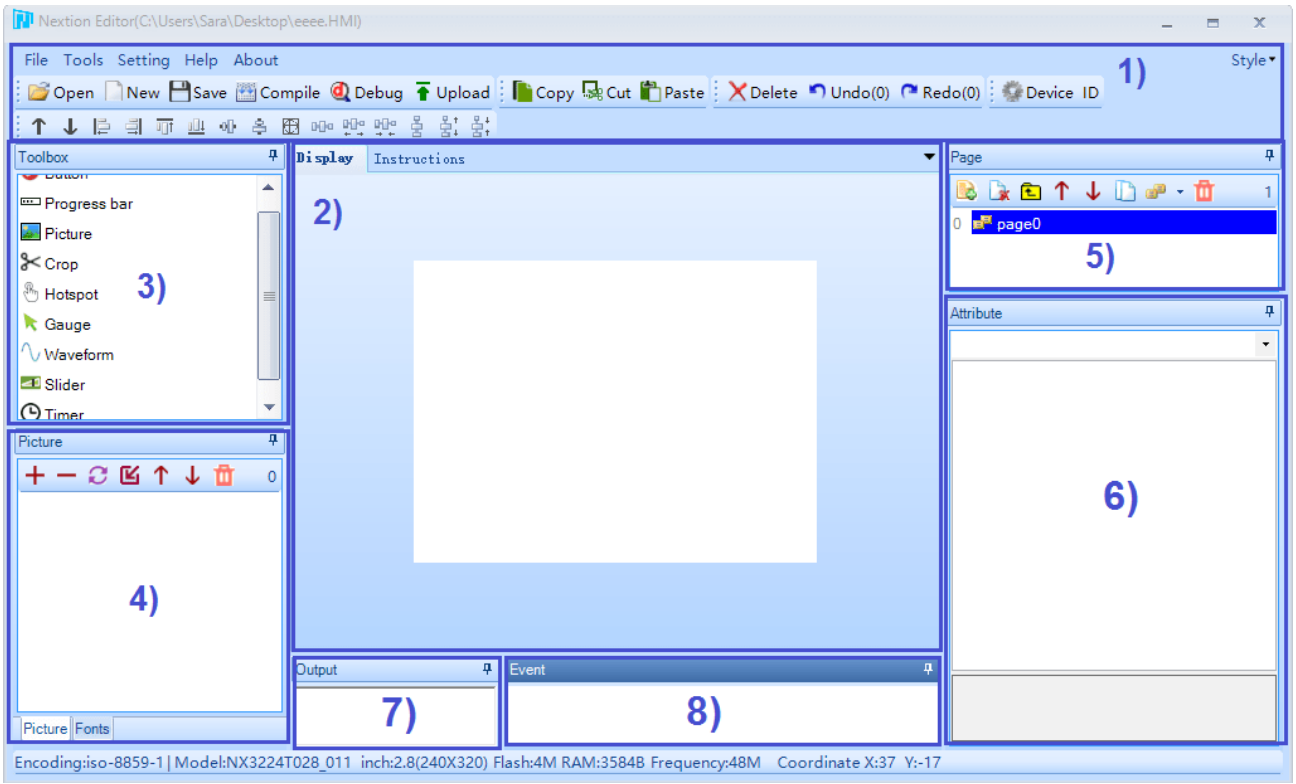


Figura 37. Interfaz gráfica de Nextion Editor
https://images.app.goo.gl/ZUJ3UJhVmb6cj2br9

Sus partes más importantes están representadas por:

1) Menú principal
2) **Canvas**: Se agregan los componentes para la interfaz gráfica.
3) **Toolbox:** Aquí se encuentra gran variedad de componentes que se pueden agregar a la interfaz como botones, barras de desplazamiento, imágenes de fondo entre otras.
4) **Picture/Fonts List:** Muestra las fuentes y las imágenes que se importan al proyecto.
5) **Attributes Area**: En esta parte se muestran los atributos de los componentes.
6) **Compiler Output Window**: Se muestran los errores ocurridos durante la compilación.
7) **Event Window:** Se agrega el código que se ejecutará cuando se cumpla el evento.

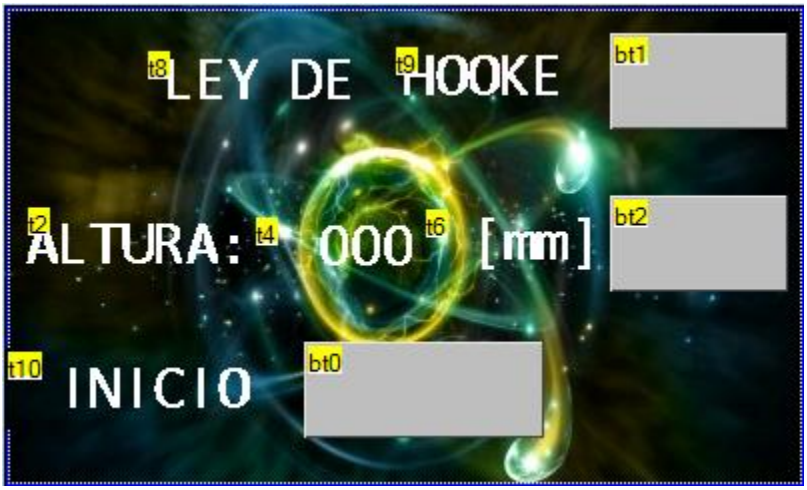


Figura 38. GUI Ley de Hooke

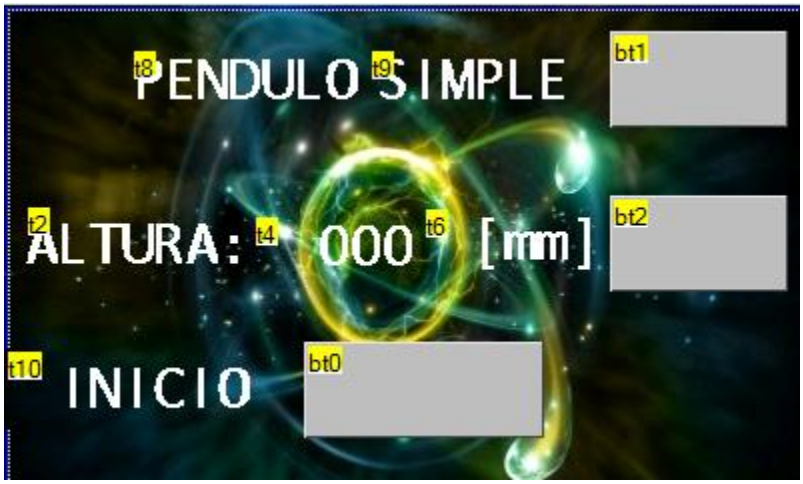


Figura 39. GUI Péndulo Simple

Figura 40. GUI Caída Libre

4.1.2. Diagrama de bloque del código implementado en Atmega de Arduino

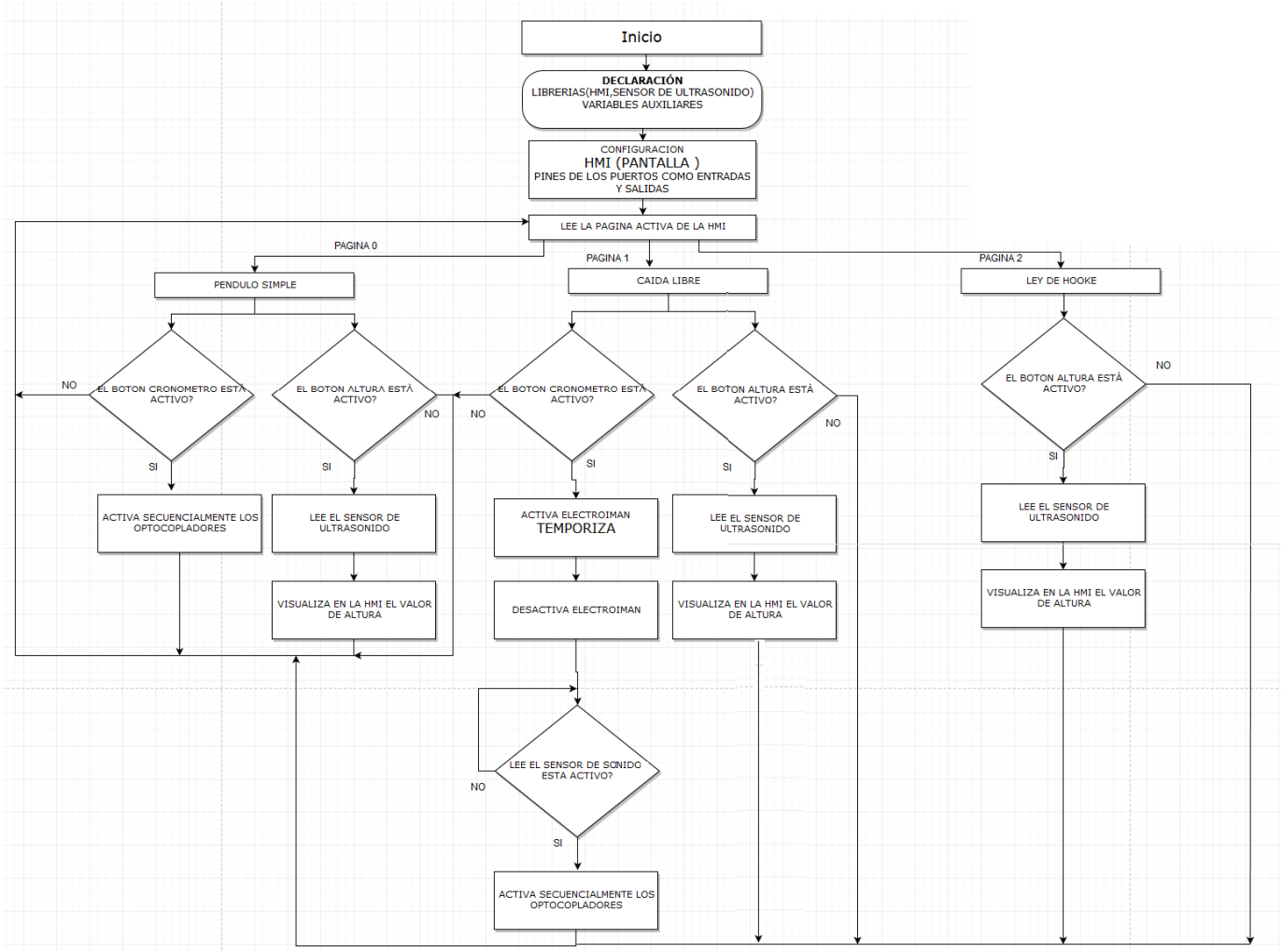


Figura 41. Diagrama de bloques del código implementado en ATMEGA

4.2. Programa Quartus II

Este es un software diseñado por Altera para el análisis y la síntesis de diseños realizados en HDL (Lenguaje de descripción de Hardware). Quartus II permite al desarrollador o desarrolladora compilar sus diseños, realizar análisis temporales, examinar diagramas RTL y configurar el dispositivo de destino con el programador (HETPRO, 2019) [15].

En la figura 40 se muestra la interfaz gráfica del programa utilizado para programar la NIOS II.

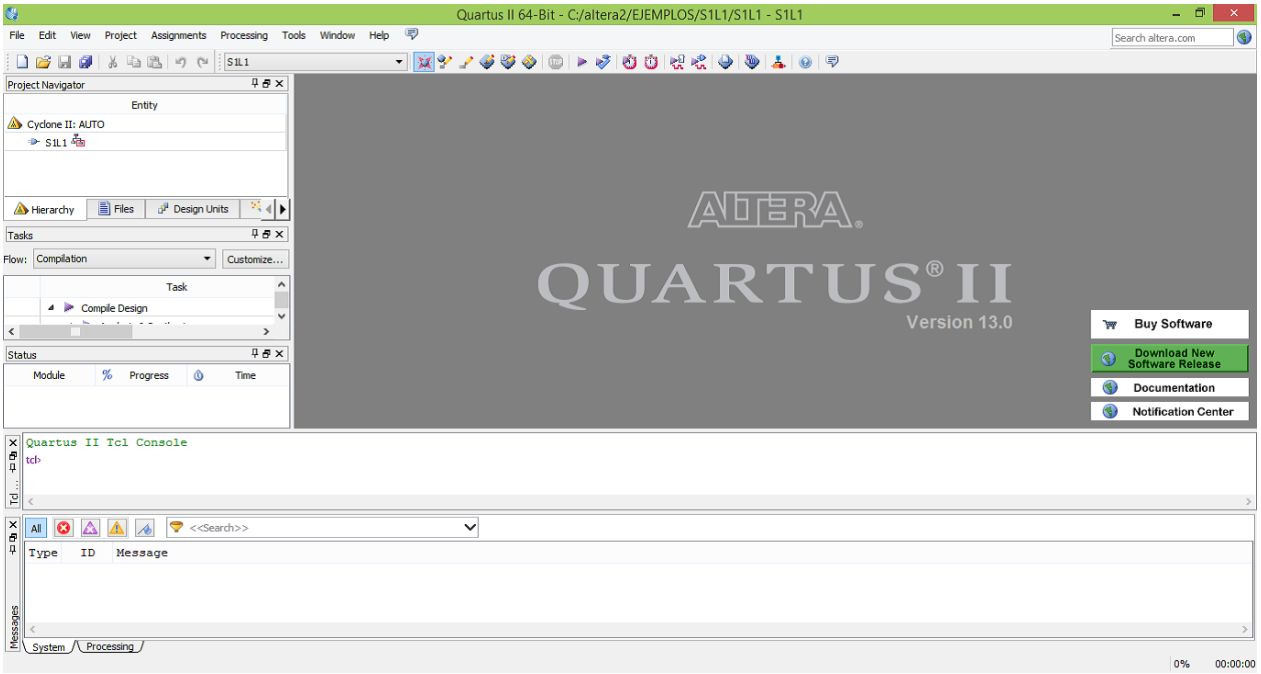


Figura 42. Interfaz gráfica QUARTUS II

https://www.studocu.com/es/document/universidade-de-vigo/tecnoloxia-electronica/trabajo-tutorial/altera-quartus-ii-herramienta-de-diseno-digital/3663866/view

4.2.1. Diagrama de bloques del código implementado en la FPGA

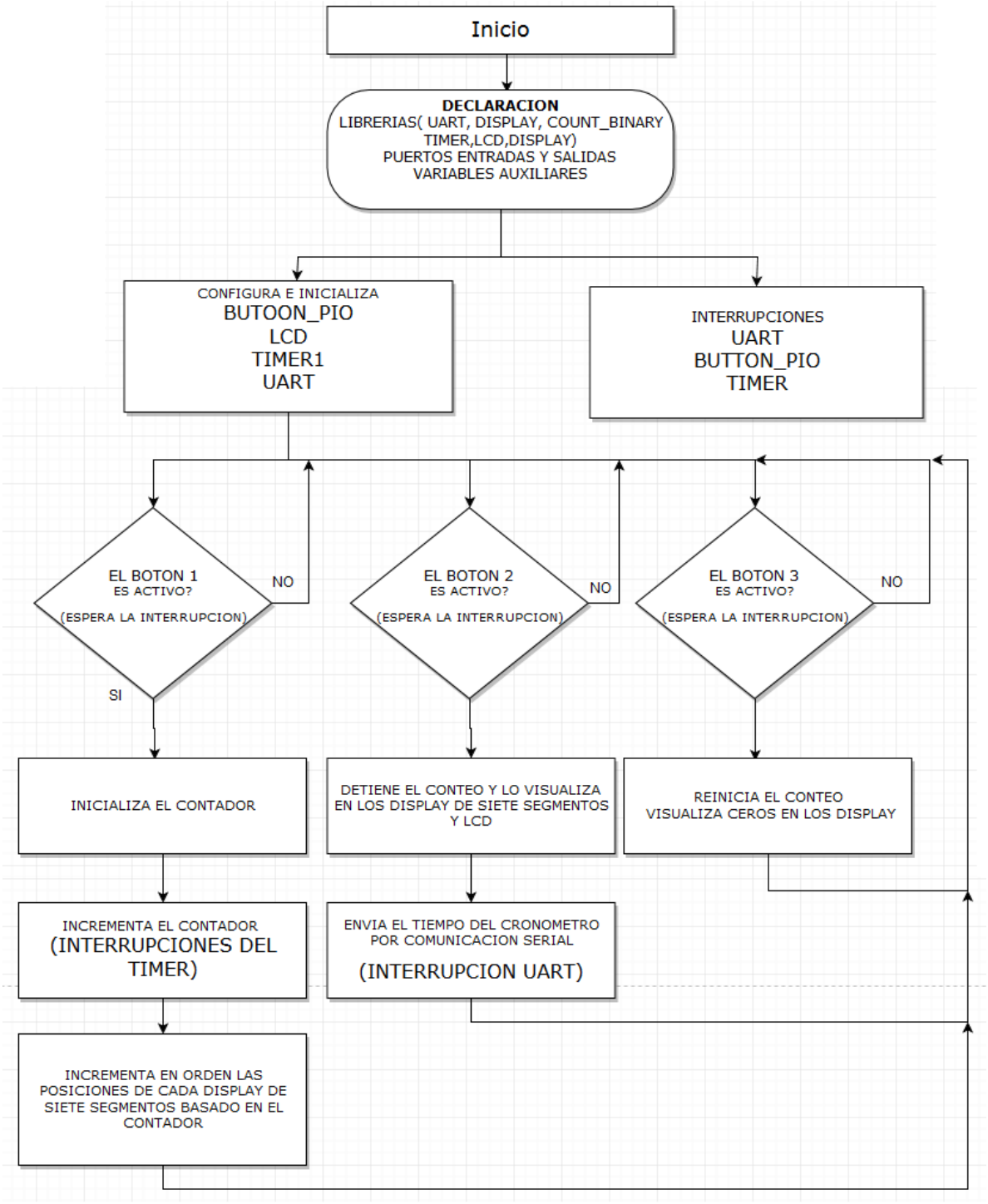


Figura 43. Diagrama de bloques del código implementado en la FPGA

CAPÍTULO 5
GUÍAS DE LAS PRÁCTICAS

5.1. Ley de Hooke

Cuando se habla de fuerzas, se tiene presente que según las leyes de Newton un cuerpo puede sufrir cambios en su posición con respecto al tiempo, pero también existen casos donde la fuerza puede deformar al objeto al que se le aplique, en este caso si se hala un resorte, este se alarga, si el resorte esta calibrado la distancia que se alarga puede usarse para medir la intensidad de la fuerza (SERWAY & JEWETT, 2008) [8].

En el siglo XVII, al estudiar los resortes y la elasticidad, el físico Robert Hooke observó que para muchos materiales la curva de esfuerzo vs. deformación tiene una región lineal. Dentro de ciertos límites, la fuerza requerida para estirar un objeto elástico, como un resorte de metal, es directamente proporcional a la extensión del resorte. A esto se le conoce como la ley de Hooke (Khan Academy, 2019) [16].

La ecuación (1) representa dicha ley.:

$$\boldsymbol{F} = -\boldsymbol{k}\,\boldsymbol{x}; \qquad (1)$$

F es la fuerza y para el sistema internacional de unidades se mide en newton (N)
k es la constante de proporcionalidad o constante del resorte y sus unidades son N/m
x es la longitud de deformación, expresada generalmente en metros.

5.1.1. Objetivos

5.1.1.1. Objetivo general

Determinar experimentalmente la ecuación que relaciona la fuerza y la deformación en resortes.

5.1.1.2. Objetivos específicos

- Encontrar la constante de elasticidad de dos resortes a través de las mediciones realizadas
- Aplicar el método de los mínimos cuadrados para lograr el ajuste a una línea recta, de los datos obtenidos en las mediciones.

5.1.2. Técnicas de Linealización

Para el caso de funciones no lineales, se trabajara en esta guía con el documento empleado por la Universidad Tecnológica de Pereira, para el análisis gráfico de una función no lineal, el cual se presenta a continuación:

Toda línea recta que resulta de unir puntos en un diagrama cartesiano, puntos que provienen de variables experimentales elevadas a la potencia 1 (la unidad), se corresponde con el método frecuentemente empleado y de garantizado éxito denominado análisis gráfico, respaldado y refinado por el método analítico denominado regresión lineal o ajuste por mínimos cuadrados.

El análisis gráfico, permite determinar primero intuitivamente y luego confirmar la relación sugerida entre dos o más variables estudiadas durante la ejecución de un experimento controlado en un laboratorio, a manera de redescubrimiento sobre las proporcionalidades que “asocian o ligan” las cantidades experimentales objeto de estudio, presentadas como hipótesis que deben someterse a verificación para comprobar su bondad o validez.

Durante un experimento, quienes lo realizan, necesitan explorar cuidadosamente que atributos, propiedades o cantidades varían para identificar, sin equívoco las variables. Deben definir entonces aquellas magnitudes Físicas que cambian o toman la categoría de variables fundamentales de la Física; estudiar de sus transformaciones y modificaciones, cómo se manifiestan las variaciones, porqué se dan, qué o quién las genera y cuáles magnitudes podrían ser variables pero que durante alguna parte o durante todo el experimento toman solo un valor, cantidades denominadas parámetros, también reconocer y usar variables frecuentes en experimentos científicos, por ejemplo: tiempo, longitud, área, superficie, masa, temperatura, intensidad de corriente, intensidad luminosa, cantidad de sustancia, velocidad, aceleración, fuerza, energía y muchas más.

Un grupo de trabajo luego de haber dispuesto y montado adecuadamente dentro de una sala experimental el equipo requerido para su labor, inicia el estudio de un fenómeno o de una práctica de laboratorio y realiza medidas cuidadosas, sobre las identificadas variables dependiente Vd y la variable independiente Vi, valores que consigna ordenadamente en tablas diseñadas y elaboradas para ese propósito.

Vi (Unidad de medida)	Vi_1	Vi_2	Vi_3	Vi_4	Vi_5	Vi_6	…	Vi_n
Vd (Unidad de medida)	Vd_1	Vd_2	Vd_3	Vd_4	Vd_5	Vd_6	…	Vd_n

Tabla 2. Cuadro genérico que contiene los posibles valores de las variables Vd y Vi medidos durante la ejecución de un experimento.

Los valores experimentales recogidos y organizados en la tabla 2, se examinan para identificar los valores extremos o limites inferior y superior de todas las medidas registradas en cada una de las **variables**, posteriormente se **evalúan en parejas los valores consecutivos para cada variable** por separado, si encuentra que, "cada valor posterior es mayor que el anterior y ese comportamiento se mantiene se tendrá una variable creciente representada así:

"Sí valor Vd_2 es > que Vd_1 ⇒ Vd es Creciente"; igualmente para la otra variable sí cada valor posterior examinado es menor que el anterior y además se conserva tal regularidad en los demás valores, entonces se tendrá una variable decreciente y en consecuencia se adopta por comodidad la siguiente notación: "Sí valor Vi_2 es < que Vi_1 ⇒ Vi es Decreciente".

Posteriormente se evalúa cómo se comportan o cambian entre sí las dos variables experimentalmente registradas, es decir, sí Vd se incrementa, entonces ¿Qué le sucede a la variable Vi? ¿También se incrementará o se decrementará?, lo cual se denota, cuando Vd↑⇒Vi? Un análisis similar permite evaluar, sí Vd decrece entonces ¿Qué le pasará a la variable Vi?, con la siguiente representación Vd↓ ⇒ Vi?; Si a través del examen preliminar se observa que cuando Vd crece, igualmente lo hace Vi , en la misma proporción, con la representación Vd↑⇒Vi↑, entonces se presume razonablemente una proporcionalidad directa entre las variables, análisis inicial simple que adquiere la categoría de **hipótesis**, la cual debe ser verificada o probada posteriormente.

Para realizar la gráfica, se utilizara la hoja de cálculo. En el caso de obtener una gráfica no lineal, se procede a utilizar el método de mínimos cuadrados

El caso más simple está asociado con la ecuación de la línea recta que mejor representa un conjunto de datos experimentales, es decir llegar a una ecuación de la forma estudiada en la primera parte de este experimento $Vd = f(Vi)$ que para análisis grafico tendrían la siguiente correspondencia:

$Vd \rightarrow$ Variable dependiente eje vertical u ordenada.
$Vi \rightarrow$ Variable independiente eje horizontal o abscisa.

Cuando se grafican variables sobre un plano, los lugares geométricos o puntos experimentales en muchas oportunidades no dejan muy claro, que al unirles reproduzcan una línea recta única, sino que muchas rectas perfectamente podrían ser representativas de los puntos dibujados; razón por la

cual surge obligadamente la pregunta ¿Cuál es la recta correcta?, en consecuencia, se explorará a continuación un método analítico, para atender la inquietud planteada.

Lo esperable es emplear un procedimiento matemático para identificar la "mejor recta" de un conjunto de puntos dado, evitar la inseguridad de un juicio subjetivo. Igualmente reconocer lo que expresa la "mejor recta", y evaluar el rigor de tal elección.

El procedimiento en cuestión toma como base el principio estadístico de los mínimos cuadrados y considera este en una aplicación restringida para encontrar una línea recta que se ajuste a los valores medidos. Se supone un conjunto de "n" valores de una variable Vd , medidos como función de la variable Vi , se restringe al caso especial de que toda la incertidumbre se limita a la dimensión Vd : esto es, los valores de Vi se conocen exactamente, o al menos, con una precisión tanto mayor que la de los valores de Vd , para poder despreciar la incertidumbre en la dimensión Vi . Si no se satisface esta condición, el tratamiento sencillo que se explica a continuación no será válido y el método requerido está fuera de este contexto.

Nuevamente la pregunta a satisfacer ahora con este procedimiento matemático es: ¿Cuál de todas las líneas en el plano Vd – Vi se escoge como la mejor, con qué criterio y que significa definir "la mejor recta"? El principio de mínimos cuadrados permite hacer esta escogencia sobre el principio de las desviaciones de los puntos en dirección vertical a partir de las posibles líneas. Sea la Línea Recta LR en la figura 41 un prospecto con la categoría de la "mejor línea recta". Obsérvese los diferentes intervalos verticales entre los puntos experimentales y la recta escogida de los cuales C_2 H_2 es típico. Se define como mejor recta aquella que minimiza la suma de los cuadrados de las desviaciones C_2 H_2 (Ec. 2)

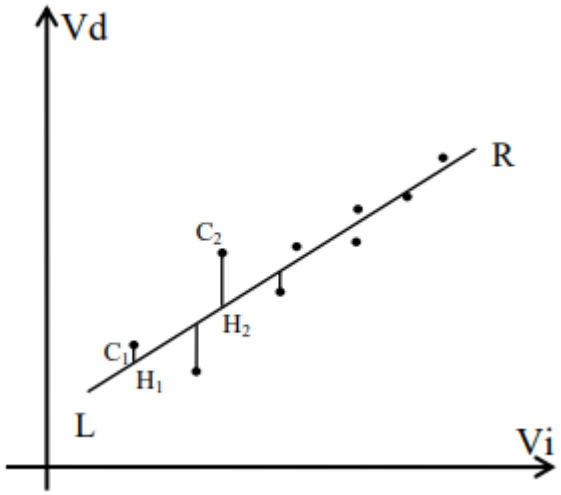


Figura 44. Ajuste de una línea recta a un conjunto de puntos por el principio de mínimos cuadrados. [19]

Se nota que un criterio sugerido como este no proporciona un camino automático hacia las respuestas "verdaderas" o "correctas" únicas. Es una opción de criterio para optimizar la trayectoria de línea propuesta entre los puntos. Se reconoce, sin embargo, ventajas sobre otras posibilidades, verbigracia cómo minimizar la potencia cúbica de los intervalos, o la primera, etc. Aunque no hace falta, en general, ocuparse directamente de la justificación lógica del principio de mínimos cuadrados tal como se aplica. Se puede probar que el procedimiento de minimizar los cuadrados de las desviaciones da lugar, en muestreos repetidos, a una menor varianza de los parámetros resultantes, como por ejemplo la pendiente, que al usar cualquier otro criterio. En consecuencia, el método brinda confianza más en los resultados obtenidos usando el principio de mínimos cuadrados, que en el caso de cualquier otro método comparable; de aquí que el uso de este principio este muy difundido. Se expresa el principio de mínimos cuadrados en forma matemática al definir que la mejor línea se corresponde con aquella que minimiza o lleva a su valor mínimo la suma del cuadrado de las desviaciones verticales.

Desviación

$$\sum_{1}^{n}(C_2H_2)^2 \qquad (2)$$

Además se debe calcular los parámetros: pendiente a y la ordenada al origen b, de esa mejo línea; considérese como la ecuación (3) que representa la mejor línea recta:

$$Vd = aVi + b \tag{3}$$

La magnitud de la desviación C_iH_i es el intervalo entre un cierto valor medido Vd_i, y el valor de Vd den ese punto, para el valor de Vd. Este valor Vd se puede calcular a partir del valor correspondiente de V_i como $aVi + b$, de modo que, si se llama δV_i, a cada diferencia, se tiene,

$$\delta Vd_i = Vd_i - (aVi + b) \tag{4}$$

El criterio de mínimos cuadrados permite calcular los valores deseados de a y b a partir de la condición representada en la ecuación (5):

$$\sum [Vd_i - aVi_i + b]^2 = Mínimo \tag{5}$$

O equivalente

$$\sum [Vd_i - aVi_i + b]^2 = Mín \tag{6}$$

Aplicando la condición mostrada en la ecuación (6) para que sea un mínimo:

$$\frac{\partial Min}{\partial a} = 0 \quad y \quad \frac{\partial Min}{\partial b} = 0$$

Un breve ejercicio algebraico permite obtener la pendiente y la ordenada al origen de la mejor recta, para ello el cuadro siguiente facilita el cálculo de las operaciones indicadas sobre las variables, en las dos columnas de la derecha. Las sumas en cada columna, visualizadas en las celdas inferiores de la tabla 3 son reemplazadas en las ecuaciones (7) y (8):

Medida N_0	Vd_i Eje vertical	Vi_i Eje horizontal	$(Vi_i)^2$	$Vi_i * Vd_i$
1				
2				
3				
4				
5				
6				
⋮				
N=	$\sum Vd_i =$	$\sum Vi_i =$	$\sum (Vi_i)^2 =$	$\sum (Vi_i * Vd_i)$

Tabla 3. Cuadro genérico para procesar datos provenientes de un experimento

$$\boldsymbol{a} = \frac{\boldsymbol{n}\sum \boldsymbol{V_{i_i}}(\boldsymbol{udm}) \cdot \boldsymbol{V_{d_i}}(\boldsymbol{udm}) - \sum \boldsymbol{V_{i_i}}(\boldsymbol{udm}) \cdot \sum \boldsymbol{V_{d_i}}(\boldsymbol{udm})}{\boldsymbol{n}\sum \left(\boldsymbol{V_{i_i}}(\boldsymbol{udm})\right)^2 - \left(\sum \boldsymbol{V_{i_i}}(\boldsymbol{udm})\right)^2} \quad (7)$$

$$\boldsymbol{b} = \frac{\sum \boldsymbol{V_{d_i}}(\boldsymbol{udm}) \cdot \sum \left(\boldsymbol{V_{i_i}}(\boldsymbol{udm})\right)^2 - \sum \boldsymbol{V_{i_i}}(\boldsymbol{udm}) \sum \boldsymbol{V_{i_i}}(\boldsymbol{udm}) \cdot \boldsymbol{V_{d_i}}(\boldsymbol{udm})}{\boldsymbol{n}\sum \left(\boldsymbol{V_{i_i}}(\boldsymbol{udm})\right)^2 - \left(\sum \boldsymbol{V_{i_i}}(\boldsymbol{udm})\right)^2} \quad (8)$$

Donde:

Vi (udm) y Vd (udm) Son los diferentes valores experimentales de las variables; (udm) representa las unidades de medida de cada variable.

Lo anterior controla el uso a veces cuestionable de la apreciación subjetiva de quien evalúa gráficamente la información experimental; ahora realizado a través del procedimiento analítico, que ofrece resultados cercanos a valores de verdad aceptables y de gran acogida por los investigadores casi en forma universal. Además, éste método contiene significado estadístico que

permite una forma próxima para el cálculo de la incertidumbre. El principio de mínimos cuadrados facilita inmediatamente valores de la desviación estándar de la pendiente y la ordenada al origen, lo que proporciona incertidumbres con significado estadístico conocido.

5.1.3. Materiales

Dos resortes

Tres masas

Mini balanza digital Sonivox VS-DWS1418

Modulo para estudio de la ley de Hooke

5.1.4. Procedimiento

- Coloque cada una de las masas en la minibalanza digital, realice tres mediciones y complete la siguiente tabla

	medida 1	medida 2	medida 3	promedio
masa 1 (kg)				
masa 2 (kg)				
masa 3 (kg)				

Tabla 4. Medición de las masas

- Conecte el equipo a la red eléctrica.
- Introduzca en la parte superior del prototipo el módulo correspondiente a la ley de Hooke.

Figura 45. Módulo ley de Hooke

- Inserte el resorte y cierre el aditamento que va a permitir tomar la lectura de la longitud natural.
- En la pantalla TÁCTIL seleccione ley de Hooke, oprimiendo el botón número 1 y a continuación pulse el botón número 2 y tome el dato que aparece en la pantalla.

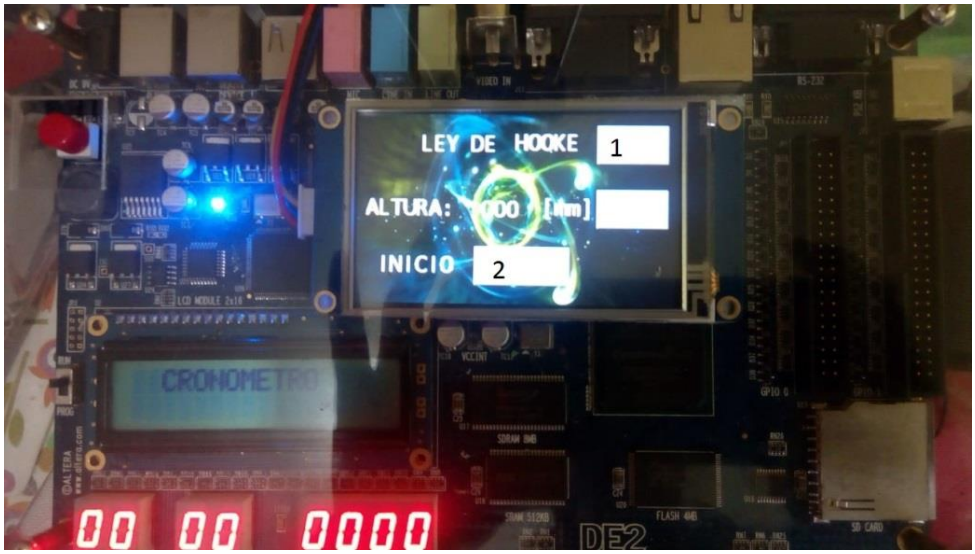


Figura 46. Pantalla Táctil – Ley de Hooke

- Quite el aditamento, coloque una de las masas, espera que se detenga las oscilaciones, cierre el aditamento y tome la lectura que aparece en la pantalla táctil (realice esto 3 veces). Repita estos pasos para cada una de las masas, cada uno de los resortes y complete la tabla 5:

Masa (g)		Deformación (cm)			Promedio deformación (cm)	Fuerza (N)
masa 1						
masa 2						
masa 3						

Tabla 5. Datos Recolectados para el resorte 1

Masa (g)		Deformación (cm)			Promedio deformación (cm)	Fuerza (N)
masa 1						
masa 2						
masa 3						

Tabla 6. Datos Recolectados para el resorte 2

- Utilizando la hoja de cálculo realice la gráfica de Fuerza vs Deformación para cada resorte
- Si la gráfica obtenida es una línea recta, halle la ecuación de la recta y compárela con la Ley de Hooke, obtenga el valor de la constante del resorte.
- Si la gráfica obtenida no es una línea recta, debe aplicar el método descrito para lograr la linealización de la recta obtenida, donde Vd es la fuerza y Vi es la deformación del resorte
- Obtenga la ecuación y el coeficiente de correlación (r) y compárelo con la expresión obtenida de la hoja de cálculo al dar clic derecho sobre la recta y elegir agregar línea de tendencia lineal y activar la casilla para ver la ecuación de la recta y el coeficiente de correlación.
- Compare los resultados y concluya

5.2. Péndulo Simple

Un péndulo simple (también llamado péndulo matemático) se define como una partícula suspendida de un punto fijo por medio de un hilo inextensible y sin peso. Al separar el péndulo de la vertical un ángulo θ, como se observa en la figura 44, la fuerza recuperadora es igual a $mgsen\theta$ y el desplazamiento a partir de la posición de equilibrio es igual a $L\theta$, donde L es la longitud del hilo y θ la medida del ángulo en radianes. (SEARS, ZEMANSKY, 2018) [7]

El movimiento por consiguiente, no es armónico, ya que la fuerza recuperadora es proporcional a $sen\theta$, mientras que la elongación lo es a θ y la fuerza recuperadora será:

$$F \approx mg\theta \approx -(mg/L)\, s \qquad (9)$$

La constante recuperadora efectiva del péndulo es $T \approx 2\pi\sqrt{(L/g)}$

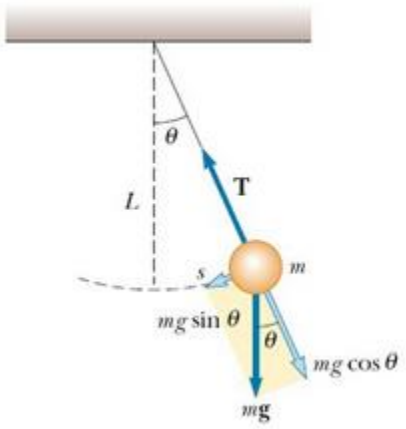


Figura 47. Fuerza sobre la masa de un péndulo simple [8]

5.2.1. Objetivo

Determinar experimentalmente el valor de la gravedad con su incertidumbre

5.2.2. Incertidumbre (Teoría)

Basándose en las guías de experimentos de física de ciencias básicas de la Universidad Tecnológica de Pereira, UTP (Pereira, 2019) [19], se puede hablar de las incertidumbres en las mediciones para los experimentos realizados con este prototipo. Se estima que se tienen como fuentes de incertidumbre las debidas a repetición en la toma de datos y la resolución del cronómetro (0,1 ms) y la resolución de la balanza (0,1g).

De acuerdo a la definición dada en las guías del experimento 2, TRATAMIENTO ESTADÍSTICO DE DATOS EXPERIMENTALES Y APLICACIÓN DEL MÉTODO GENERAL PARA EL CÁLCULO DE INCERTIDUMBRE DE MEDICIÓN (MEDIDAS DIRECTAS), se tienen los dos métodos principales para evaluar las incertidumbres:

Tipo A: Método de evaluación de una incertidumbre estándar mediante el análisis estadístico de una serie de observaciones, se estima basándose en mediciones repetidas obtenidas del mismo proceso de medición, es decir que la incertidumbre tipo A se obtiene a partir de las mediciones realizadas en el laboratorio y se calcula con la desviación estándar de las mediciones dividido por la raíz cuadrada del número de mediciones

$$\mu_A = \frac{\sigma}{\sqrt{n}} \tag{10}$$

donde σ es la Desviación Estándar y n el número de datos.

Para la Tipo B, se trabaja con el caso 2, es decir, por resolución y en este caso por instrumentos de medición análogos

$$\mu B2 = \frac{RESOLUCIÓN}{2\sqrt{3}} \tag{11}$$

Para el cálculo de la incertidumbre estándar combinada, se utiliza la siguiente ecuación:

$$\mu C = \sqrt{(\mu A)^2 + (\mu B2)^2} \tag{12}$$

Por ultimo para el cálculo de la incertidumbre extendida, $UE = k * \mu C$, para k que es el factor de cobertura, el cual está dado por e l número de grados de libertad del sistema de medición, se va a considerar como $k = 1{,}96$, con un número infinito de grados de libertad y un 95% como nivel de confianza.

5.2.3. Materiales

- Modulo para estudio del péndulo simple
- Péndulo
- Transportador

5.2.4. Procedimiento

- Conecte el equipo a la red eléctrica
- Introduzca en la parte superior del prototipo el módulo correspondiente a Péndulo Simple

Figura 48. Soporte para Péndulo Simple

- En la pantalla TÁCTIL seleccione péndulo simple, oprimiendo el botón número 1 y a continuación pulse el botón número ON y tome el dato que aparece en la pantalla
- Saque el hilo del péndulo y llévelo hasta la posición de 55 cm, suéltelo y oprima iniciar, espere que haga el recorrido y escriba el dato que se ve en la pantalla, repita esto por lo menos 5 veces para realizar el promedio.
- Recoja el hilo y tome lecturas como en el paso anterior recortando 5 cm cada vez, hasta llegar 10 cm, complete la siguiente tabla:

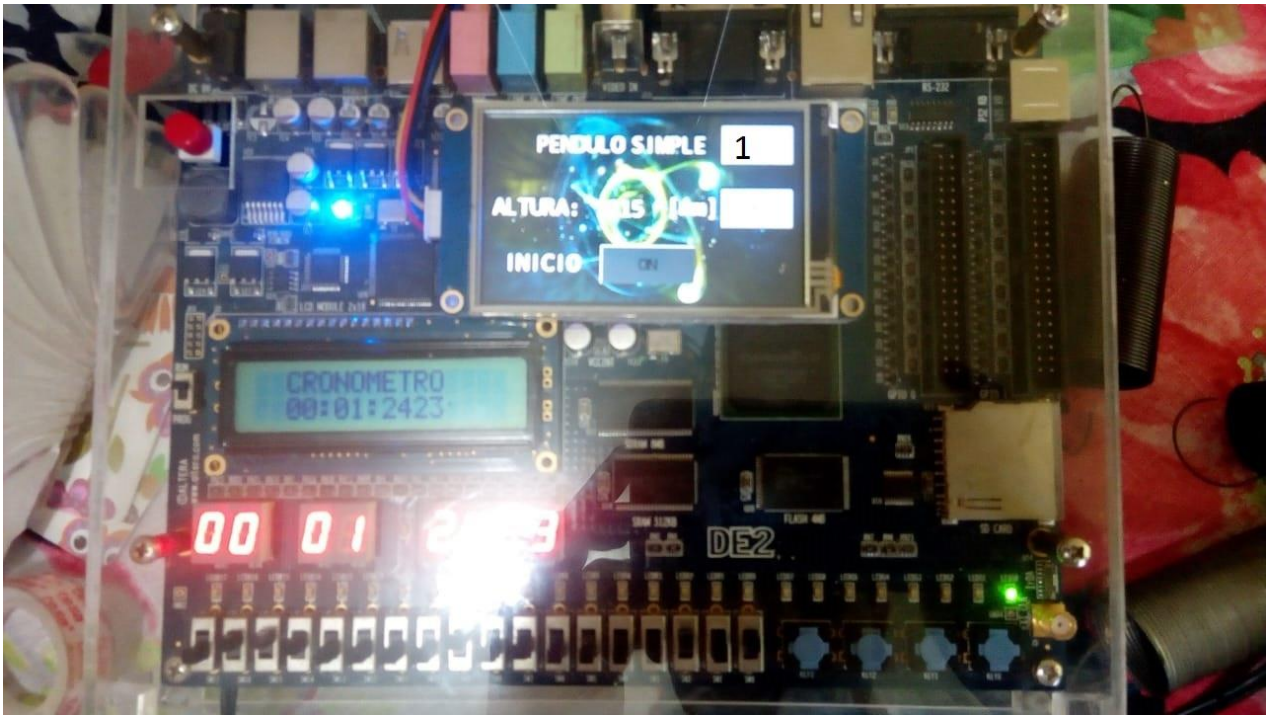


Figura 49. Pantalla Táctil - Péndulo Simple

TABLA DE DATOS PENDULO SIMPLE						
DATO	t de una oscilación (s)	t promedio	Período T =tiempo (s)/numero oscilaciones	Longitud (m)	L promedio	T^2
1						
2						
3						
4						
5						

Tabla 7. Datos péndulo simple

- Realice la gráfica de Periodo vs Longitud, ¿qué forma tiene la gráfica?
- Linealice la gráfica utilizando el método de mínimos cuadrados.
- Según su concepto, ¿funciona esta linealización?
- Realice la gráfica de $T2\ vs\ L$.

- Utilizando la ayuda de hoja de cálculo al dar clic derecho sobre la recta y elegir agregar línea de tendencia lineal y activar la casilla para ver la ecuación de la recta y el coeficiente de correlación.
- Compare el resultado de la pendiente con el valor de $4\pi^2/g$, y despeje el valor de g.
- Calcule la incertidumbre en la pendiente y reporte el valor de la gravedad como $g \pm \Delta g$

5.3. Caída Libre

Un objeto lanzado hacia arriba y uno lanzado hacia abajo experimenta la misma aceleración que un objeto que se deja caer desde el reposo. Una vez que estén en caída libre, todos los objetos tienen una aceleración hacia abajo, igual a la aceleración de caída libre. Este movimiento se representa mediante las ecuaciones 13, 14, 15 y 16.

Si se desprecia la resistencia del aire y se supone que la aceleración en caída libre no varía con la altitud, entonces el movimiento vertical de un objeto que cae libremente es equivalente al movimiento en una dimensión con aceleración constante. Por tanto, pueden aplicarse las ecuaciones cinemáticas para aceleración constante.

$$v = vo - gt \tag{13}$$

$$h - ho = \frac{1}{2}(v + vo)\, t \tag{14}$$

$$h - ho = vo\, t - \frac{1}{2} gt^2 \tag{15}$$

$$v^2 = vo^2 - 2g\,(h - ho) \tag{16}$$

Advirtiendo que el signo negativo para la aceleración ha está incluido en las expresiones.

Si la altura inicial h y la velocidad inicial son cero, las expresiones simplificadas quedan representadas por las ecuaciones (17) y (18):

$$h = -\frac{1}{2} gt^2 \tag{17}$$

$$v^2 = -2g\,h \tag{18}$$

5.3.1. Objetivos

- Determinar experimentalmente el valor de la gravedad con su incertidumbre
- Analizar el comportamiento de un objeto que cae desde determinada altura

5.3.2. Materiales

- Módulo para estudio de caída libre
- Un balín

5.3.3. Procedimiento

- Conecte el equipo a la red eléctrica
- Introduzca en la parte superior del prototipo el módulo correspondiente a Caída Libre

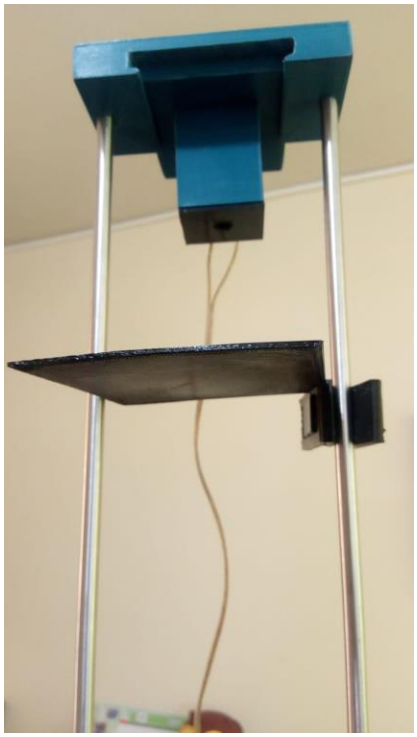


Figura 50. Caída Libre

- En la pantalla TÁCTIL seleccione caída libre, oprimiendo el botón número 1 h a continuación pulse el botón número 2 h tome el dato de altura que aparece en la pantalla.

Figura 51. Pantalla Táctil - Caída Libre

- Ubique el balín en la posición del electroimán, en diez segundos se desactiva el electroimán h el objeto cae, registre para la misma altura cinco datos del tiempo h escríbalos en la siguiente tabla 8:

Serie 1			Serie 2			Serie 3			Serie 4			Serie 5	
$h(m)$	$t(s)$		$h(m)$	$t(s)$		$h(m)$	$t(s)$		$h(m)$	$t(s)$		$h(m)$	$t(s)$

Tabla 8. Altura vs Tiempo - Caída Libre

- Saque los promedios de los datos registrados.
- Utilizando la hoja de cálculo.

- Realice una gráfica de las alturas promedio (hi) como una función del promedio del tiempo (t). ¿Qué tipo de grafica obtienen?.
- Linealize la gráfica obtenida, qué puede decir de este proceso.
- Realice una gráfica de h/t como una función de t.
- Realice una gráfica de h en función de t^2.
- Que puede decir de la pendiente encontrada.
- Calcule la incertidumbre en la pendiente h reporte el valor de la gravedad como $g \pm \Delta g$.

CAPÍTULO 6

RESULTADOS EXPERIMENTALES

6.1. Ley de Hooke

Datos obtenidos a partir de la medición de cinco masas diferentes.

HOOKE										
Medidas	DESPLAZAMIENTO 1 (mm)	MASA 1 (g)	DESPLAZAMIENTO 2 (mm)	MASA 2 (g)	DESPLAZAMIENTO 3 (mm)	MASA 3 (g)	DESPLAZAMIENTO 4 (mm)	MASA 4 (g)	DESPLAZAMIENTO 5 (mm)	MASA 5 (g)
1	306	9,2	281	18,9	269	29,1	245	35,8	231	42,9
2	305	9,1	281	19,0	270	29,0	246	35,8	232	42,8
3	306	9,1	281	19,0	269	29,0	246	35,8	231	43,0
4	306	9,1	282	19,1	270	29,1	246	35,8	231	42,9
5	305	9,1	281	19,0	270	28,9	246	35,9	231	42,9
6	306	9,1	282	19,0	270	29,0	246	35,8	232	42,9
7	306	9,1	281	19,0	270	29,0	246	35,6	231	42,8
8	305	9,1	281	19,1	269	29,1	245	35,6	231	42,9
9	305	9,1	280	19,0	270	29,0	246	35,6	231	42,9
10	305	9,1	281	19,0	270	29,0	246	35,6	231	42,9

Tabla 9. Datos experimentales Ley de Hooke

PROMEDIOS	
MASA (g)	DESPLAZAMIENTO (mm)
305,5	9,11
281,1	19,01
269,7	29,02
245,8	35,73
231,2	42,89

Tabla 10. Promedio de masa

PROMEDIOS	
FUERZA (N)	DESPLAZAMIENTO (m)
2,994E+00	9,110E-03
2,755E+00	1,901E-02
2,643E+00	2,902E-02
2,409E+00	3,573E-02
2,266E+00	4,289E-02

Tabla 11. Promedio de Fuerzas

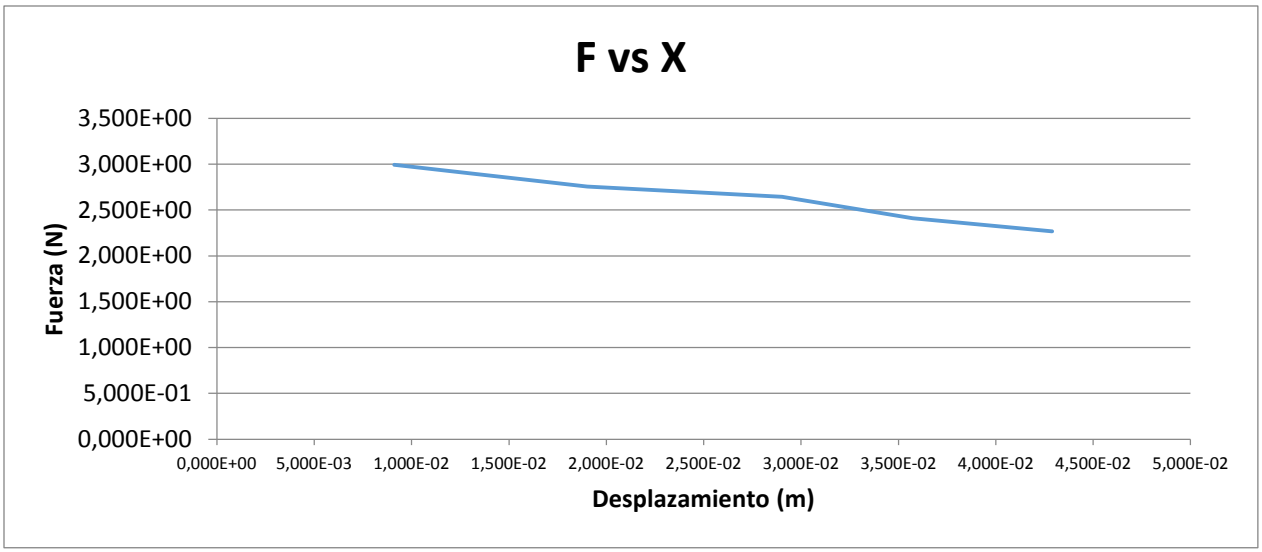


Figura 52. Grafica Fuerza vs Deformación

6.2. Péndulo Simple

Datos obtenidos a partir de la medición de cinco longitudes diferentes.

PENDULO SIMPLE										
Medidas	LONGITUD 1 (mm)	TIEMPO 1 (s)	LONGITUD 2 (mm)	TIEMPO 2 (s)	LONGITUD 3 (mm)	TIEMPO 3 (s)	LONGITUD 4 (mm)	TIEMPO 4 (s)	LONGITUD 5 (mm)	TIEMPO 5 (s)
1	570	1,5095	445	1,3472	383	1,2427	286	1,0965	162	0,8312
2	570	1,507	445	1,3438	383	1,242	286	1,0943	162	0,8268
3	570	1,5073	445	1,3436	383	1,2466	286	1,0934	162	0,8227
4	570	1,5033	445	1,3442	383	1,2385	286	1,093	162	0,8192
5	570	1,506	445	1,3467	383	1,2402	286	1,0976	162	0,8176
6	570	1,5035	445	1,3472	383	1,2386	286	1,0966	162	0,8252
7	570	1,5126	445	1,3476	383	1,2381	286	1,0952	162	0,8222
8	570	1,5064	445	1,3463	383	1,2352	286	1,0935	162	0,821
9	570	1,5089	445	1,3463	383	1,2352	286	1,0994	162	0,8189
10	570	1,5053	445	1,3437	383	1,2342	286	1,0982	162	0,8256

Tabla 12. Datos experimentales Péndulo Simple

T (s)	L (m)
1,51698	0,570
1,33566	0,445
1,24213	0,383
1,08277	0,286
0,81304	0,162

Tabla 13. Datos Longitud vs Periodo

T^2 (s^2)	L(m)
2,30122832	0,570
1,783987636	0,445
1,542886937	0,383
1,172390873	0,286
0,661034042	0,162

Tabla 14. Datos T^2 vs L

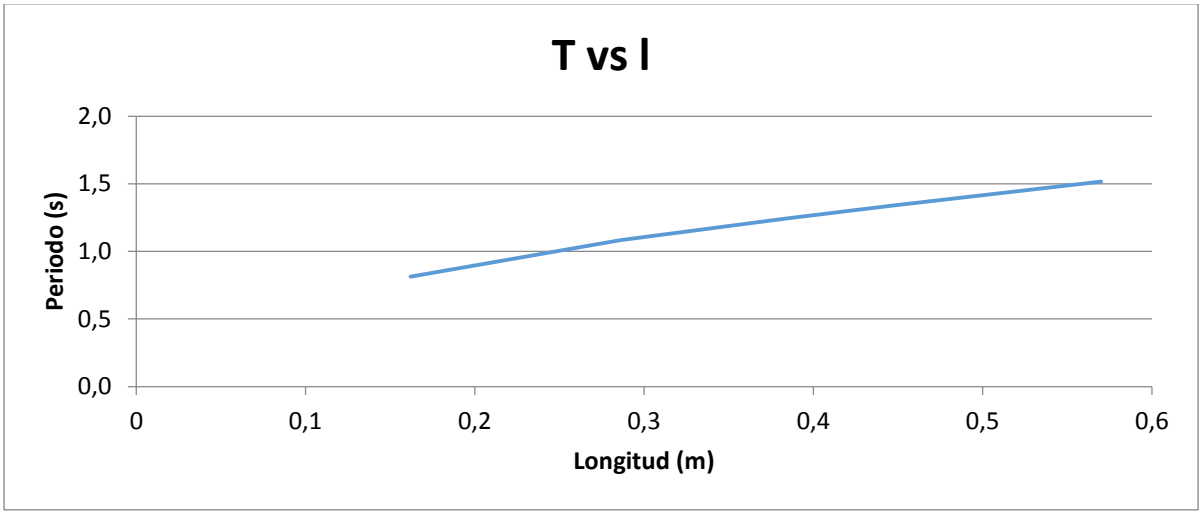


Figura 53. Gráfica Periodo vs Longitud

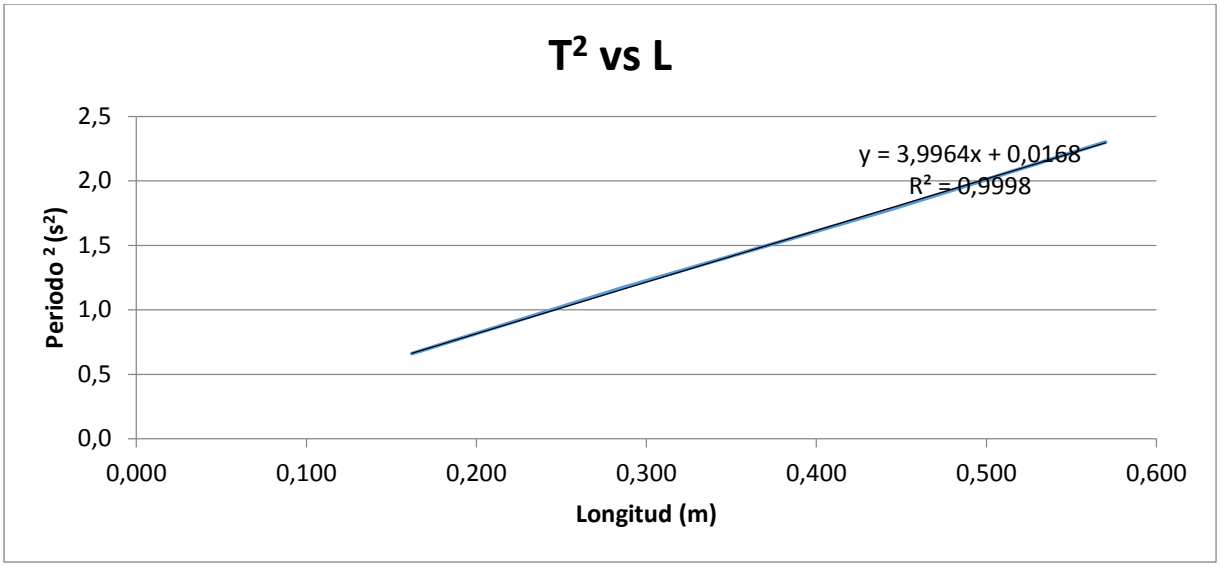


Figura 54. Gráfica T^2 vs L

6.2.1. Valor de la gravedad

desvest (m/s^2)	0,23998
µA (m/s^2)	0,023998
µB (m/s^2)	0,028868
µc (m/s^2)	0,03754
µE (m/s^2)	0,073578
g (m/s^2) =	9,878495
g=(9,878495±0,073578) m/s^2	

Tabla 15. Valor experimental de la gravedad

6.3. Caída Libre

Datos obtenidos a partir de la medición de cinco alturas diferentes.

CAIDA LIBRE										
Medidas	LONGITUD 1 (mm)	TIEMPO 1 (s)	LONGITUD 2 (mm)	TIEMPO 2 (s)	LONGITUD 3 (mm)	TIEMPO 3 (s)	LONGITUD 4 (mm)	TIEMPO 4 (s)	LONGITUD 5 (mm)	TIEMPO 5 (s)
1	637	0,3499	527	0,3352	469	0,3236	382	0,2849	250	0,2438
2	637	0,3621	528	0,3381	468	0,3166	381	0,2849	251	0,2381
3	636	0,3613	527	0,3349	469	0,3243	382	0,2853	250	0,2408
4	637	0,3628	528	0,3392	468	0,3249	382	0,2879	250	0,2382
5	637	0,3529	528	0,3344	468	0,3199	382	0,2845	250	0,2403
6	636	0,3628	527	0,3354	469	0,3178	382	0,2879	250	0,2445
7	637	0,3645	528	0,3347	469	0,3181	381	0,2746	250	0,238
8	637	0,3615	527	0,3366	469	0,3182	382	0,2761	250	0,2395
9	637	0,3627	528	0,3348	468	0,3206	382	0,2868	250	0,2466
10	637	0,3737	527	0,3374	468	0,3239	381	0,2855	251	0,2471

Tabla 16. Datos experimentales Caída Libre

y (m)	t (s)
0,6492	0,36142
0,5682	0,33607
0,5196	0,32079
0,3995	0,28384
0,2902	0,24169

Tabla 17. Datos Altura vs Tiempo

y (m)	t^2 (s^2)
0,6492	0,130624416
0,5682	0,112943045
0,5196	0,102906224
0,3995	0,080565146
0,2902	0,058414056

Tabla 18. Datos Y vs T^2

6.3.1. Valor de la gravedad

desvest (m/s^2)	0,239980224
µA (m/s^2)	0,023998022
µB (m/s^2)	0,028867513
µc (m/s^2)	0,037539824
µE (m/s^2)	0,073578056
g (m/s^2) =	10,1268
g=(10,0506±0,073578) m/s^2	

Tabla 19. Valor experimental de la gravedad

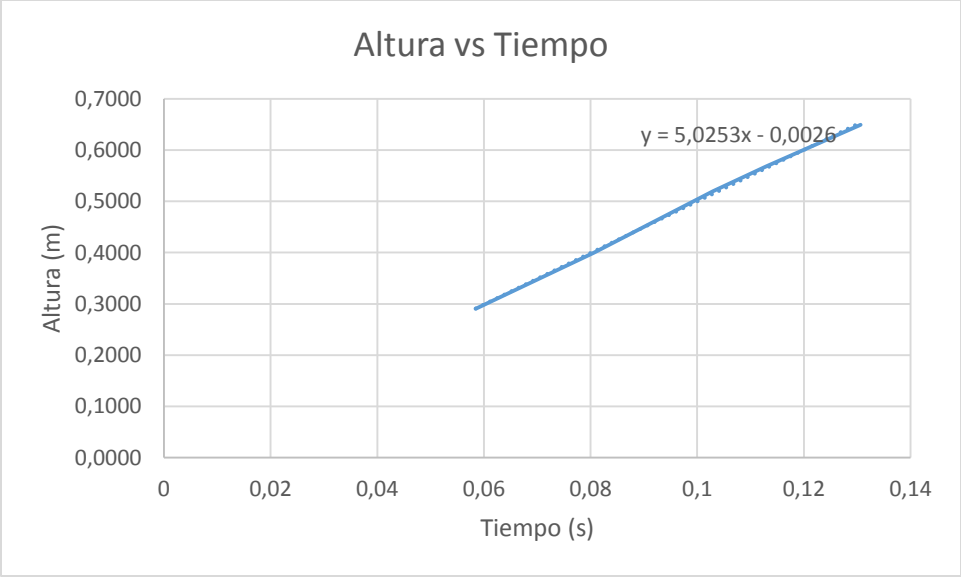


Figura 55. Gráfica Altura vs Tiempo

CONCLUSIONES Y RECOMENDACIONES

El prototipo basado en un dispositivo FPGA ofrece ventajas en aspectos como procesamiento de información, implementación de algoritmos, programación interna de un procesador, interface de comunicación, que reunidas en un solo sistema potencializa el desarrollo de las prácticas

El prototipo diseñado, gracias a su estructura modular, permite una facilidad en su utilización, cuenta además con una buena precisión en los datos tomados, todo esto hace que se puedan realizar comprobaciones de manera eficiente en relación con las prácticas tradicionales de la experimentación física, para los tres modelos utilizados.

Se comprobó que los datos de mediciones entregados por el prototipo, permiten un buen cálculo para cada una de las variables físicas estudiadas, en el caso del péndulo y caída libre se obtuvo un valor cercano al valor teórico considerado de la gravedad.

La programación de la FPGA usando lenguaje VHDL es más dispendiosa en términos de la descripción del hardware externo asociado al dispositivo. Por eso se implementa um procesador NIOS que permite configurar fácilmente el hardware adicional que está allí presente como los sensores usados.

Se destaca la alta velocidad de procesamiento debido al reloj interno de la tarjeta de desarrollo, esto es, si se quiere hacer una comparación exhaustiva con otro tipo de sistemas embebidos como son los microcontroladores. De esta forma, se muestra la eficiencia de esta clase de arquitecturas programables que aunque son más costosas ofrece múltiples interfaces de entrada/salida y captura de los datos procesados.

Inicialmente hubo muchas dificultades en la concepción del modelo, ya que no se disponían de los elementos necesarios para manipular las piezas de acrílico, doblarlo o cortarlo. Se obtuvo el apoyo de la red TecnoParque del Sena Pereira, donde se elaboraron los primeros modelos.

El prototipo construido en el desarrollo de este proyecto constituye una versión inicial al procesamiento de señales usando FPGA, por lo que está sujeto a nuevos desarrollos con otro tipo de interfaz de comunicación como son el puerto Ethernet para transmisión de datos a un archivo plano que pueda ser tratado mediante una hoja de cálculo y posteriormente su gráfica.

ANEXO I. Código del módulo ATMEGA de Arduino

```
#include "HCSR04.h"
#include <avr/interrupt.h>
#include <avr/io.h>

HCSR04 Sensor1(8,9);
const int buttonPin2 = 10;     // the number of the pushbutton pin
const int buttonPin = 11;      // the number of the pushbutton pin
const int ledPin =  13;        // the number of the LED pin

// variables will change:
int buttonState = 0;           // variable for reading the pushbutton status

float DistanceCm = 0;
int data[256];
unsigned long time1;          //Declara un variable del mismo tipo que la que devuelve micros()
unsigned long time2;       //Declara un variable del mismo tipo que la que devuelve micros()
int k = 0;
int i = 0;
int max_v = 0;
float f = 0;
uint32_t val;

int variable1=0;
int variable2=0;
int variable3=0;
int contador=0;
int promedio=0;
```

```
int entrada1 = 10;
int entrada2 = 11;
int pendulosimple=0;
int ayudante=0;
//float duration, distance; // duracion y distancia (tipo flotante)
float distancia;
long tiempo;
float temp3=0;
float temdistancia=0;
float temp4=0;
int umid;
int umid1;
int hey=0;
char buffer1[10] = {0};
char buffer2[10] = {0};
char buffer3[10] = {0};
char buffer4[10] = {0};
uint32_t dual_state;
uint32_t dual_state1;
uint32_t dual_state2;
#include "NexButton.h"
#include "NexText.h"
#include "Nextion.h"
NexText txt_umid  = NexText(0, 6, "t4");
NexText txt_umid1  = NexText(1, 6, "t4");
NexText txt_umid2  = NexText(2, 6, "t4");
NexDSButton bt0   = NexDSButton(0, 4, "bt0");
NexDSButton bt1   = NexDSButton(1, 4, "bt0");
NexDSButton bt2   = NexDSButton(2, 4, "bt0");
```

```
NexDSButton bt3   = NexDSButton(0, 9, "bt2");
NexDSButton bt4   = NexDSButton(1, 9, "bt2");
NexDSButton bt5   = NexDSButton(2, 9, "bt2");

char buffer[10] = {0};

NexTouch *nex_Listen_List[] =
{
   &bt0,
   &bt1,
   &bt2,
   &bt3,
   &bt4,
   &bt5,
   NULL
};

void bt0PopCallback(void *ptr)
{
  digitalWrite(4 , HIGH);              // poner el Pin en HIGH
  delayMicroseconds(500);              // esperar un segundo
  digitalWrite(4 , LOW);               // poner el Pin en LOW
  delayMicroseconds(500);              // esperar un segundo          // esperar un segundo
  digitalWrite(6 , HIGH);              // poner el Pin en HIGH
  delayMicroseconds(500);              // esperar un segundo          // esperar un segundo
  digitalWrite(6 , LOW);               // poner el Pin en HIGH
  delayMicroseconds(500);              // esperar un segundo          // esperar un segundo
  variable1=0;
   NexButton *btn = (NexButton *)ptr;
   memset(buffer, 0, sizeof(buffer));
```

```
    btn->getText(buffer, sizeof(buffer));
    if (strcmp(buffer,"ON"))
    {
       strcpy(buffer, "ON");
    }
    else
    {
       strcpy(buffer, "OFF");
       variable1=0;
       ayudante=1;
       delay(20);
       digitalWrite(4 , HIGH);                  // poner el Pin en HIGH
       delayMicroseconds(500);                  // esperar un segundo
       digitalWrite(4 , LOW);                   // poner el Pin en LOW
       delayMicroseconds(500);
    }
    btn->setText(buffer);
}

void bt1PopCallback(void *ptr)
{
  digitalWrite(4 , HIGH);               // poner el Pin en HIGH
  delayMicroseconds(500);               // esperar un segundo
  digitalWrite(4 , LOW);                // poner el Pin en LOW
  delayMicroseconds(500);               // esperar un segundo            // esperar un segundo

    NexButton *btn = (NexButton *)ptr;
    memset(buffer, 0, sizeof(buffer));
    btn->getText(buffer, sizeof(buffer));
    if (strcmp(buffer,"ON"))
```

```
    {
      strcpy(buffer, "ON");
      variable1=1;
      ayudante=0;
    }
    else
    {
       strcpy(buffer, "OFF");
       variable1=0;
       ayudante=1;
        delay(20);
         digitalWrite(4 , HIGH);                // poner el Pin en HIGH
         delayMicroseconds(500);                // esperar un segundo
         digitalWrite(4 , LOW);                 // poner el Pin en LOW
         delayMicroseconds(500);
   }
    btn->setText(buffer);

}
void bt2PopCallback(void *ptr)
{
   digitalWrite(4 , HIGH);               // poner el Pin en HIGH
   delayMicroseconds(500);               // esperar un segundo
   digitalWrite(4 , LOW);                // poner el Pin en LOW
   delayMicroseconds(500);              // esperar un segundo            // esperar un segundo

    ayudante=0;

    NexButton *btn = (NexButton *)ptr;
    memset(buffer, 0, sizeof(buffer));
```

```
btn->getText(buffer, sizeof(buffer));
if (strcmp(buffer,"ON"))
{
    strcpy(buffer, "ON");
    digitalWrite(7 , HIGH);   // Suelto la bola
    variable1=2;
    ayudante=0;
    delay(5000);
    digitalWrite(7 , LOW);   // Suelto la bola

    digitalWrite(6 , HIGH);   // poner el Pin en HIGH
    delayMicroseconds(500);
    digitalWrite(6 , LOW);   // poner el Pin en HIGH
    delayMicroseconds(500);
    digitalWrite(5 , HIGH);   // poner el Pin en HIGH
    delayMicroseconds(500);
    digitalWrite(5 , LOW);    // poner el Pin en LOW
    delayMicroseconds(500);

}
else
{
   strcpy(buffer, "OFF");
   variable1=0;
   ayudante=1;
    delay(20);
     digitalWrite(4 , HIGH);                // poner el Pin en HIGH
     delayMicroseconds(500);                // esperar un segundo
     digitalWrite(4 , LOW);                 // poner el Pin en LOW
     delayMicroseconds(500);
```

```
    }
    btn->setText(buffer);

}
void bt3PopCallback(void *ptr)
{
  digitalWrite(4 , HIGH);     // poner el Pin en HIGH
  delayMicroseconds(500);   // esperar un segundo
  digitalWrite(4 , LOW);      // poner el Pin en LOW
  delayMicroseconds(500);  // esperar un segundo              // esperar un segundo

  variable1=0;

    NexButton *btn = (NexButton *)ptr;
    memset(buffer, 0, sizeof(buffer));
    btn->getText(buffer, sizeof(buffer));
    if (strcmp(buffer,"ON"))
    {
       strcpy(buffer, "ON");
        DistanceCm = Sensor1.ping(58,615);    // Formula: uS / 58 = centimeters ; 500 = offset HC-
SR0; 615 = offset US-015
       distancia =  (DistanceCm *10)-80;//-50;

        if(distancia>0){
             k=distancia;
             }
           if(distancia==0){
             umid=k;
             }
```

```
            else
            {
            umid=distancia;
           }
           txt_umid.setText(buffer3);
           memset(buffer3, 0, sizeof(buffer3));
           itoa(umid, buffer3, 10);

   }
   else
   {
      strcpy(buffer, "OFF");
      variable1=0;
      ayudante=1;
      delay(20);
      digitalWrite(4 , HIGH);   // poner el Pin en HIGH
      delayMicroseconds(500);                  // esperar un segundo
      digitalWrite(4 , LOW);                  // poner el Pin en LOW
      delayMicroseconds(500);

   }
   btn->setText(buffer);

}
void bt4PopCallback(void *ptr)
{
  digitalWrite(4 , HIGH);              // poner el Pin en HIGH
  delayMicroseconds(500);             // esperar un segundo
  digitalWrite(4 , LOW);               // poner el Pin en LOW
  delayMicroseconds(500);             // esperar un segundo            // esperar un segundo
```

```
  variable1=0;

   NexButton *btn = (NexButton *)ptr;
   memset(buffer, 0, sizeof(buffer));
   btn->getText(buffer, sizeof(buffer));
   if (strcmp(buffer,"ON"))
   {
      strcpy(buffer, "ON");
       DistanceCm = Sensor1.ping(58,615);   // Formula: uS / 58 = centimeters ; 500 = offset HC-
SR0; 615 = offset US-015
      distancia =  (DistanceCm *10)-122;     //-50;

       if(distancia>0){
            k=distancia;
            }
          if(distancia==0){
            umid=k;
            }
            else
            {
           umid=distancia;
          }
          txt_umid.setText(buffer3);
          memset(buffer3, 0, sizeof(buffer3));
          itoa(umid, buffer3, 10);

   }
   else
   {
```

```
      strcpy(buffer, "OFF");
      variable1=0;
      ayudante=1;
      delay(20);
      digitalWrite(4 , HIGH);                  // poner el Pin en HIGH
      delayMicroseconds(500);                  // esperar un segundo
      digitalWrite(4 , LOW);                   // poner el Pin en LOW
      delayMicroseconds(500);

   }
   btn->setText(buffer);

 }
 void bt5PopCallback(void *ptr)
{
  digitalWrite(4 , HIGH);   // poner el Pin en HIGH
  delayMicroseconds(500);              // esperar un segundo
  digitalWrite(4 , LOW);               // poner el Pin en LOW
  delayMicroseconds(500);              // esperar un segundo          // esperar un segundo

  variable1=0;

   NexButton *btn = (NexButton *)ptr;
   memset(buffer, 0, sizeof(buffer));
   btn->getText(buffer, sizeof(buffer));
   if (strcmp(buffer,"ON"))
   {
      strcpy(buffer, "ON");
       DistanceCm = Sensor1.ping(58,615);   // Formula: uS / 58 = centimeters ; 500 = offset HC-
SR0; 615 = offset US-015
```

```
    distancia =  (DistanceCm *10)-80;         //-50;

     if(distancia>0){
           k=distancia;
           }
          if(distancia==0){
           umid=k;
           }
           else
           {
           umid=distancia;
          }
          txt_umid.setText(buffer3);
          memset(buffer3, 0, sizeof(buffer3));
          itoa(umid, buffer3, 10);

}
else
{
    strcpy(buffer, "OFF");
    variable1=0;
    ayudante=1;
    delay(20);
    digitalWrite(4 , HIGH);                  // poner el Pin en HIGH
    delayMicroseconds(500);                  // esperar un segundo
    digitalWrite(4 , LOW);                   // poner el Pin en LOW
    delayMicroseconds(500);

}
btn->setText(buffer);
```

```
 }

void setup(void){

  nexInit();
  bt0.attachPop(bt0PopCallback, &bt0);
  bt1.attachPop(bt1PopCallback, &bt1);
  bt2.attachPop(bt2PopCallback, &bt2);
  bt3.attachPop(bt3PopCallback, &bt3);
  bt4.attachPop(bt4PopCallback, &bt4);
  bt5.attachPop(bt5PopCallback, &bt5);
   pinMode(7 , OUTPUT);  //definir pin como salida
  pinMode(6 , OUTPUT);  //definir pin como salida
  pinMode(5 , OUTPUT);  //definir pin como salida
  pinMode(4 , OUTPUT);  //definir pin como salida
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
  pinMode(buttonPin2, INPUT);

  pendulosimple=0;
    //bt0.attachPop(bt0PopCallback, &bt0); //callback para o evento de release do botão btnNext
   // bt0.attachPop(bt1PopCallback, &bt1); //callback para o evento de release do botão btnBack
   // bt1.attachPop(bt2PopCallback, &bt2); //callback para o evento de release do slider
    }

void loop(){
```

```
if(i==0){
  digitalWrite(4 , HIGH);                 // poner el Pin en HIGH
  delayMicroseconds(500);                 // esperar un segundo
  digitalWrite(4 , LOW);                  // poner el Pin en LOW
  delayMicroseconds(500);                 // esperar un segundo          // esperar un segundo
  digitalWrite(6 , HIGH);                 // poner el Pin en HIGH
  delayMicroseconds(500);                 // esperar un segundo          // esperar un segundo
  digitalWrite(6 , LOW);                  // poner el Pin en HIGH
  delayMicroseconds(500);                 // esperar un segundo          // esperar un segundo

  i=1;

  }
  nexLoop(nex_Listen_List);

  if(variable1==1){

     buttonState = digitalRead(buttonPin);
     if((buttonState==HIGH)&& (ayudante==0)){
      ayudante=1;
      }
    if((buttonState==LOW)&& (ayudante==1)){

      digitalWrite(6 , HIGH);   // poner el Pin en HIGH
      delayMicroseconds(500);
      digitalWrite(6 , LOW);   // poner el Pin en HIGH
      delayMicroseconds(500);
      digitalWrite(5 , HIGH);   // poner el Pin en HIGH
      delayMicroseconds(500);
      digitalWrite(5 , LOW);    // poner el Pin en LOW
```

```
 delayMicroseconds(500);
 ayudante=2;
}
if((buttonState==HIGH)&& (ayudante==2)){
 ayudante=3;
 }
 if((buttonState==LOW) && (ayudante==3)){
   ayudante=4;
   }
 if((buttonState==HIGH)&& (ayudante==4)){
   ayudante=5;
 }
 if((buttonState==LOW)&& (ayudante==5)){

   digitalWrite(6 , HIGH);   // poner el Pin en HIGH
   delayMicroseconds(500);
   digitalWrite(6 , LOW);   // poner el Pin en HIGH
   delayMicroseconds(500);
   digitalWrite(5 , HIGH);   // poner el Pin en HIGH
   delayMicroseconds(500);
   digitalWrite(5 , LOW);    // poner el Pin en LOW
   delayMicroseconds(500);
                // esperar un segundo
   ayudante=0;
  variable1=10;

  }
}
if(variable1==2){
```

```
    buttonState = digitalRead(buttonPin2);

    if((buttonState==HIGH)&& (ayudante==0)){

      digitalWrite(6 , HIGH);   // poner el Pin en HIGH
      delayMicroseconds(500);
      digitalWrite(6 , LOW);   // poner el Pin en HIGH
      delayMicroseconds(500);
      digitalWrite(5 , HIGH);   // poner el Pin en HIGH
      delayMicroseconds(500);
      digitalWrite(5 , LOW);    // poner el Pin en LOW
      delayMicroseconds(500);
                  // esperar un segundo
      ayudante=0;
      variable1=10;
     }
    }
}
```

ANEXO 2. Código implementado en la FPGA

```
/*bps 115200*/
#include "altera_avalon_uart_regs.h"
#include "alt_types.h"
#include "sys/alt_irq.h"
#define  UART_BASE  0x00681000
#define  UART_IRQ   2
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "system.h"
#include "sys/alt_irq.h"
#include "altera_avalon_pio_regs.h"
#include "count_binary.h"
#include "lcd.h"
#include <sys/unistd.h>
#include <io.h>
#include "altera_avalon_timer_regs.h"
int pos=0;
  int year,month,day,hour,minute,second,minutos;
  unsigned long sum;
  char date[16];
  char time[17];
  char text1[16]="  CRONOMETRO";
  int year1=7;
  int year2=0;
  int year3=0;
  int year4=2;
  int month1=2;
```

```
  int month2=1;
  int day1=9;
  int day2=0;
  int ayudante1=0;
  int ayudante2=0;
  int ayudante3=0;
  int ayudante4=0;
  int pendulo=0;
  int resortecaida=0;
  int contador=0;
  int bandera=0;
  int cronometro=0;
  int programar=0;

  int hour4,hour3,hour2,hour1,minute2,minute1,second2,second1;
  unsigned int screenflag;
  hour=0;minute=0;minutos=0;second=0;year=2006;month=1;day=1;

void LCD_Init()
{
    lcd_write_cmd(LCD_16207_0_BASE,0x38);
  usleep(2000);
  lcd_write_cmd(LCD_16207_0_BASE,0x0C);
  usleep(2000);
  lcd_write_cmd(LCD_16207_0_BASE,0x01);
  usleep(2000);
  lcd_write_cmd(LCD_16207_0_BASE,0x06);
  usleep(2000);
  lcd_write_cmd(LCD_16207_0_BASE,0x80);
  usleep(2000);
```

```
}
void LCD_Show_Text(char* Text)
{
  int i;
  for(i=0;i<strlen(Text);i++)
  {
    lcd_write_data(LCD_16207_0_BASE,Text[i]);
    usleep(2000);
  }
 }

 void LCD_Line1()
 {
  lcd_write_cmd(LCD_16207_0_BASE,0x80);
  usleep(2000);
 }

 void LCD_Line2()
 {
  lcd_write_cmd(LCD_16207_0_BASE,0xC0);
  usleep(2000);
 }

static void timer1_init(void);
int i = 0;
volatile int edge_capture;

void Uart_send(unsigned char data)
{
    alt_u16 status;
```

```
    status=IORD_ALTERA_AVALON_UART_STATUS(UART_BASE);
    while(!(status&0x0040))//
        status=IORD_ALTERA_AVALON_UART_STATUS(UART_BASE);
    IOWR_ALTERA_AVALON_UART_TXDATA(UART_BASE,data);
}

void Uart_send_n(unsigned char *ptr)
{
    while(*ptr)
    {
      Uart_send(*ptr);
      ptr++;
    }
    Uart_send(0x0a);//
}

int Uart_receive(void)
{
    alt_u16 status;
    int temp;
    status=IORD_ALTERA_AVALON_UART_STATUS(UART_BASE);
    while(!(status&0x0080))//
    status=IORD_ALTERA_AVALON_UART_STATUS(UART_BASE);
    temp=IORD_ALTERA_AVALON_UART_RXDATA(UART_BASE);
    return temp;
}
void Uart_ISR(void * context,alt_u32 id)
{
    unsigned char temp;
    temp=IORD_ALTERA_AVALON_UART_RXDATA(UART_BASE);
```

```
    switch(temp)
    {
        case '0':
       IOWR(SEG7_DISPLAY_BASE,0,0x00000000);break;
        case '1':
       IOWR(SEG7_DISPLAY_BASE,0,0x11111111);break;

       case '2':
       IOWR(SEG7_DISPLAY_BASE,0,0x22222222);break;
        case '3':
       IOWR(SEG7_DISPLAY_BASE,0,0x33333333);break;
        case '4':
       IOWR(SEG7_DISPLAY_BASE,0,0x44444444);break;
        case '5':
       IOWR(SEG7_DISPLAY_BASE,0,0x55555555);break;
        case '6':
       IOWR(SEG7_DISPLAY_BASE,0,0x66666666);break;
        case '7':
       IOWR(SEG7_DISPLAY_BASE,0,0x77777777);break;
        case '8':
       IOWR(SEG7_DISPLAY_BASE,0,0x88888888);break;
        case '9':
       IOWR(SEG7_DISPLAY_BASE,0,0x99999999);break;
    }

void Uart_init()
{
    IOWR_ALTERA_AVALON_UART_CONTROL(UART_BASE, 0x80);//
    IOWR_ALTERA_AVALON_UART_STATUS(UART_BASE, 0x0);//
    alt_irq_register(UART_IRQ,0,Uart_ISR);
```

```
}

static void  handle_button_interrupts(void *context,alt_u32 id)
{
   volatile int * edge_capture_ptr=(volatile int *)context;


*edge_capture_ptr=IORD_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE);

   IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE,0);
}

static void init_button_pio()
{
   void *edge_capture_ptr=(void*)&edge_capture;

   IOWR_ALTERA_AVALON_PIO_IRQ_MASK(BUTTON_PIO_BASE,0xf);

   IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE,0x0);

   alt_irq_register(BUTTON_PIO_IRQ,edge_capture_ptr,handle_button_interrupts);
}

int main(void)
{
   char text1[16]="  CRONOMETRO";
   init_button_pio();
   LCD_Init();
   LCD_Line1();
   LCD_Show_Text(text1);
```

```
 Uart_init();
 Uart_send_n("Isabella,I love u!");
//Uart_send_n(64);

 timer1_init();

init_button_pio();
while(1)
{
   switch(edge_capture)
   {
      case 0x08:
      IOWR_ALTERA_AVALON_PIO_DATA(LED_GREEN_BASE, resortecaida);
      bandera=1;
      IOWR(SEG7_DISPLAY_BASE,0,0X00000000);
      hour=0;minute=0;minutos=0;second=0;month=1;day=1;
      ayudante1=0;
      programar=1;
      cronometro=1;
      if (ayudante4==1){
         if (pendulo==1){
            contador=3;
             }
         }
      break;

      case 0x04:

      if (programar==1){
         cronometro=0;
```

```
 }
IOWR_ALTERA_AVALON_PIO_DATA(LED_GREEN_BASE, resortecaida);
ayudante3=1;
ayudante4=0;

if ((bandera==1) && (programar==1)) {
   contador=contador+1;
   bandera=0;
   if (contador==4){
      pendulo=1;
      resortecaida=0;
      contador=0;
     }
 else {
   resortecaida=1;
   pendulo=0;
    }

 if (contador>>4){
   contador=0;
    }
   }

break;

case 0x02:

if (cronometro==1){
   programar=0;
 }
```

```
 IOWR_ALTERA_AVALON_PIO_DATA(LED_GREEN_BASE, resortecaida);
 ayudante4=1;
if ((ayudante1==0) && (ayudante3==1)){
 ayudante1=1;
  }
  if ((ayudante1==1) && (resortecaida==1) && (ayudante3==1)){
     ayudante2=ayudante2+1;
     if (ayudante2==2){
        ayudante1=0;
        ayudante2=0;
        LCD_Line2();
        LCD_Show_Text(time);
        Uart_send_n(time);
       }
   }

  if ((ayudante1==1) && (pendulo==1) && (ayudante3==1)){
     ayudante2=ayudante2+1;
     if (ayudante2==4){
     ayudante1=0;
     ayudante2=0;
     LCD_Line2();
     LCD_Show_Text(time);
     Uart_send_n(time);
     }
  }
 contador=0;
 ayudante3=0;
 break;
```

```
            case 0x01:
               IOWR(SEG7_DISPLAY_BASE,0,0x44444444);
            break;
          }
       }
}
static void  handle_button_interrupts(void *context,alt_u32 id);
static void ISR_timer1(void *context, alt_u32 id)
{
     if (ayudante1==1){
       IOWR_ALTERA_AVALON_PIO_DATA(LED_GREEN_BASE, pendulo);
       i++;
   if(second<99) second++;
       else{
         second=0;
        if(minute<99) minute++;
          else{
          minute=0;
          if(hour<59) hour++;
             else{
             hour=0;
             if(minutos<59) minutos++;
             else{
             minutos=0;
             }
             }
          }
         }
          hour4=minutos/10;
```

```
                hour3=minutos%10;
                hour2=hour/10;
                hour1=hour%10;
                minute2=minute/10;
                minute1=minute%10;
                second2=second/10;
                second1=second%10;
                time[0]=' ';time[1]=' ';
                time[2]=hour4+0x30;time[3]=hour3+0x30;
                time[4]=':';
                time[5]=hour2+0x30;time[6]=hour1+0x30;
                time[7]=':';time[8]=minute2+0x30;
                time[9]=minute1+0x30;
                time[10]=second2+0x30;time[11]=second1+0x30;
                time[12]=' ';time[13]=' ';
                time[14]=' ';time[15]=' ';

IOWR_ALTERA_AVALON_PIO_DATA(SEG7_DISPLAY_BASE,hour4*0x10000000+hour3
*0x1000000+hour2*0x100000+hour1*0x10000+minute2*0x1000+minute1*0x100+second2*0x
10+second1*0x1);
            }
            if(i == 8)
               i = 0;

               IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_1_BASE, 0x00);
        }

        void timer1_init(void)
        {
           IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_1_BASE, 0x00);
```

```
    //Timer1=25000000/5000000=0.5s
    IOWR_ALTERA_AVALON_TIMER_PERIODL(TIMER_1_BASE,10000);
    IOWR_ALTERA_AVALON_TIMER_PERIODH(TIMER_1_BASE,10000 >> 16);
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_1_BASE, 0x07);
    alt_irq_register(TIMER_1_IRQ, (void *)TIMER_1_BASE, ISR_timer1);
}
```

BIBLIOGRAFÍA

[1] Aberga Farro, P. (2014). DISEÑO E IMPLEMENTACIÓN DE UNA PLATAFORMA VIRTUAL PARA SISTEMAS EMBEBIDOS UTILIZANDO UN PROCESADOR SOFT-CORE EMBEBIDO EN UN FPGA. Lima, Perú.

[2] Altera, C. (2006). *Development and Education Board. User Manual.*

[3] Anon. (1995). *New Design Tools for the FPGA User Trade.* Londres: Journal of Electronic Enginnering.

[4] BROWN, S., & VRANESIC, Z. (2005). *Fundamentals of Digital Logic with VHDL Design.* New York: McGraw-Hill.

[5] Castillo Morales, E. (Junio de 2008). PROTECCIÓN DE LA PROPIEDAD INTELECTUAL DE CIRCUITOS DIGITALES PARA LA SÍNTESIS DE DESCRIPCIONES DE ALTO NIVEL. Granada, España.

[6] Chan, P., & Mourad, S. (1994). *Digital Design Using Field Programmable Gate Arrays.* Prentice Hall.

[7] SEARS, ZEMANSKY. (2018). *Física Universitaria.* Pearson.

[8] SERWAY, R., & JEWETT, J. (2008). *Física para Ciencias e Ingeniería.* CENGAGE Learning.

[9] Sisterna , C. (2003). FIELD PROGRAMMABLE GATE ARRAYS. San Juan, San Juan, Argentina.

[10] WAKERLY, J. (1994). *Digital Design: Principles & Practices.* McGraw-Hill.

WEBGRAFÍA

[11] BANNER. (2019). *RS Component*. Obtenido de https://es.rs-online.com/web/p/sensores-fotoelectricos/8601480/

[12] CDMX. (2019). *CDMX Electrónica*. Obtenido de https://www.cdmxelectronica.com/nuevos_productos/sensores/keyes/modulo-ky-037-sensor-de-sonido/

[13] *COMPIC Electrónica*. (2018). Obtenido de https://www.compic.es/bipolar/377-optoacoplador-pc-817.html

[14] *Geek Factory*. (2018). Obtenido de https://www.geekfactory.mx/tutoriales/tutoriales-arduino/sensor-ultrasonico-hc-sr04-y-arduino/

[15] *HETPRO*. (2019). Obtenido de https://hetpro-store.com/TUTORIALES/como-instalar-quartus-ii-y-crear-un-proyecto-nuevo/

[16] *Khan Academy*. (2019). Obtenido de https://es.khanacademy.org/science/physics/work-and-energy/hookes-law/a/what-is-hookes-law

[17] *Microchip*. (2015). Obtenido de http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

[18] *NEXTION*. (2019). Obtenido de https://nextion.itead.cc/

[19] Pereira, U. T. (2019). *TRATAMIENTO ESTADÍSTICODE DATOSEXPERIMENTALES Y APLICACIÓN DEL MÉTODO GENERAL PARA EL CÁLCULO DE INCERTIDUMBREDE MEDICIÓN*. Obtenido de https://media.utp.edu.co/facultad-ciencias-basicas/archivos/EXP-2-trat-estd-datos-exp-medirectas.pdf

[20] PROMETEC. (2019). *PROMETEC*. Obtenido de https://www.prometec.net/sensor-sonido-ky038/#modal

[21] *Random Nerd Tutorials*. (2019). Obtenido de https://randomnerdtutorials.com/nextion-display-with-arduino-getting-started/

[22] REDATEL. (2018). *REDATEL*. Obtenido de https://www.redatel.net/html/a-que-se-refiere-la-proteccion-ip66.html

[23] *RS Components Ltd*. (2019). Obtenido de https://es.rs-online.com/web/generalDisplay.html?id=designspark/designspark-mechanical

[24] Ruiz Carbajal, R. (Octubre de 2012). *UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA*. Obtenido de http://jupiter.utm.mx/~tesis_dig/11608.pdf

[25] Texas Instruments. (2019). Obtenido de http://www.ti.com/lit/ds/symlink/max232.pdf