ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"DISEÑO Y CONSTRUCCIÓN DE UN PROTOTIPO BASADO EN FPGA Y SISTEMAS EMBEBIDOS, PARA CONTROLAR UN DISPOSITIVO DE MEDICIÓN DE NO2 EN CUERPOS DE AGUA Y ANALIZAR SUS RESULTADOS"

INFORME DE PROYECTO INTEGRADOR

Previo a la obtención del Título de:

INGENIERO/A EN TELECOMUNICACIONES

CASTILLO QUINTO EDGAR JAVIER

BRAVO LINO JORGE ALFREDO

GUAYAQUIL – ECUADOR

AÑO: 2019

AGRADECIMIENTOS

Agradezco a mis padres Edgar y Yolanda por siempre estar ahí cuando más lo necesitaba y estaba a punto de darme por vencido, por ser un ejemplo a seguir y darme sus consejos. A mis amigos que de alguna u otra manera me ayudaron en la elaboración de este proyecto. A los ingenieros Victor Arce y Nathaly Sanchez por su apoyo en todo momento y ayudarme a cumplir esta meta. A mi amigo y compañero de tesis Jorge por su trabajo duro y sacar el proyecto adelante.

Edgar Javier Castillo Quinto

Agradezco a DIOS por mostrarme siempre el camino a seguir ante las situaciones difíciles. A mi madre Efigenia Lino que soporto tanto y se esforzó para ayudarme a cumplir esta meta; también a mi padre José Bravo que siempre estuvo aconsejándome y apoyándome en todas las etapas de mi vida, tratando de darme ánimos cada día para seguir adelante. A Evelyn Arreaga quien hasta la escritura de este documento es mi compañera de vida y madre de mi pequeño hijo, por ser un apoyo, no solo emocional sino también económico, por saber entender los momentos difíciles que pasé y siempre tratar de cumplir todos mis caprichos, motivándome a salir adelante, y, por hacerme saber que se sentía orgullosa con cada paso que daba. A Kerly Ochoa, Christian Sacarelo, Rodrigo Castro, Daniel Ochoa, quienes conforman Byodinamics por confiar en mí y brindar su apoyo para la realización de este proyecto. A mi amigo y compañero de proyecto integrador, Edgar Castillo, porque fuimos un gran equipo de trabajo en cada proyecto que participamos, también, porque supo brindarme su apoyo en los momentos difíciles de mi vida. Por último, a mi pequeño hijo Elian Bravo que con solo una sonrisa y un abrazo lograba darme fuerzas y motivarme para levantarme cada día, aunque no tuviera la energía para hacerlo.

Jorge Alfredo Bravo Lino

DEDICATORIA

A mi familia que en todo momento estuvieron ahí para brindarme su apoyo y a cada uno de mis amigos que confiaron en mí.

Edgar Javier Castillo Quinto

A mi madre deseo que siempre me apoyo, por lo que deseo se sienta orgullosa de mí, al igual que mi padre. También al resto de mi familia y amigos que confiaron en mí y que alguna vez dieron su granito de arena para que yo pudiera cumplir esta meta.

Jorge Alfredo Bravo Lino

DECLARACIÓN EXPRESA

"La responsabilidad y la autoría del contenido de este Trabajo de Titulación, nos corresponde exclusivamente; Edgar Castillo y Jorge Bravo otorgamos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Edgar Javier Castillo Quinto

Jorge Alfredo Bravo Lino

RESUMEN

Los sistemas industriales de medición de parámetros y control de procesos han utilizado PLCs durante muchos años. Sin embargo, con el desarrollo de nuevas tecnologías, su integración con nuevos fabricantes resulta compleja, lo cual incrementa el costo de ingeniería.

La empresa ECRobotics ha desarrollado un sistema de medición de concentración de nitritos en el agua que utiliza un PLC como sistema de control, y, el presente trabajo presenta una propuesta para cambiar el controlador del sistema de medición por una tarjeta de desarrollo con características más flexibles que permita escalar sus funcionalidades.

Para efectuar el cambio del controlador, se estudiaron los detalles de la arquitectura del dispositivo y se realizó una evaluación para la elección de una tarjeta de desarrollo (FPGA) que cumpla con estos requerimientos. Para el control de los actuadores del dispositivo, se desarrolló una tarjeta electrónica de comunicación. Además, un programa embebido fue diseñado e implementado para el control del dispositivo de medición y los actuadores.

Finalmente, se realizaron pruebas de comunicación entre la tarjeta de desarrollo y la tarjeta electrónica, y, se ajustaron parámetros de calibración para garantizar el correcto funcionamiento del dispositivo.

Palabras Clave: Controlador, componentes, escalabilidad, FPGA, concentración de nitritos.

ABSTRACT

Industrial systems of parameters measurement and control process have used PLCs for many years. However, with the development of new technologies, its integration with new manufacturers results as a complex process. This makes the engineering work more expensive. The ECRobotics enterprise has developed a measurement system of nitrites concentration in water that uses a PLC in it.

This work shows a proposal to change the controller of the measurement device by an electronic development board with more flexible characteristics that make its functionalities escalate.

To make the controller change come true, the details of the architecture of the device were studied. An evaluation was done to choose a development board (FPGA) that accomplishes all the requirements. For the control of the device actuators, a communication electronic board was developed. What is more, an embedded program was designed and implemented for the control of the measurement device and its actuators.

Finally, a test of communication between the development board and the electronic board was done. Using those results, the calibration parameters were adjusted to secure the correct performance of the device.

Keywords: Controller, components, device, development board, FPGA, electronic card, measurement system

ÍNDICE

DEDICATORIA 4
RESUMEN I
ABSTRACT II
ABREVIATURAS V
SIMBOLOGÍA VII
ÍNDICE DE FIGURAS VIII
íNDICE DE TABLAS X
CAPÍTULO 1 1
1. INTRODUCCIÓN 1
1.1 Descripción del Problema 1
1.2 Justificación del Problema 2
1.2.1 Justificación Técnica 2
1.2.2 Justificación Económica 3
1.2.3 Justificación Financiera 4
1.3 Objetivos 4
1.3.1 Objetivo general 4
1.3.2 Objetivos Específicos 4
1.4 Estado del Arte 5
1.5 Alcance 6
1.6 Metodología 7
CAPÍTULO 2 9
2. MARCO TEÓRICO 9
2.1 Tarjeta lógica programable FPGA 9
2.2 Características de una FPGA 9
2.2.1 Arquitectura Programable 9
2.2.2 Desarrollo acelerado 9
2.2.3 Bajo costo de operación 9
2.2.4 Integración de hardware 10
2.3 Aplicaciones 10
2.3.1 Sistema de inteligencia artificial 10
2.3.2 Radio definido por software 10
2.3.3 Seguridad 11
2.5. Módulo de expansión de E/S digitales MicroLogix 1100 15
CAPÍTULO 3 20

3. DISEÑO DE LA SOLUCIÓN 20
3.1. Funcionamiento del dispositivo de medición de NO_2 20
3.2. Proceso de medición de nitritos 22
3.3. Esquema de interfaces electrónicas 24
3.4. Diseño del sistema digital programado en Quartus para la interconexión entre el FPGA y el sistema operativo del mismo. 31
3.4.1. Bloque Generador de PWM 32
3.4.2. Instanciación HPS 34
3.4.3. Sistema Digital Completo 37
3.5. Diseño del software controlador 39
3.6. Análisis de Datos 42
CAPÍTULO 4 43
4. ANÁLISIS DE RESULTADOS 43
4.1. IMPLEMENTACIÓN DE INTERFACES ELECTRÓNICAS ..... **¡Error! Marcador no definido.**
4.2. IMPLEMENTACIÓN DEL ALGORITMO DE CONTROL . **¡Error! Marcador no definido.**
4.3. ANÁLISIS DE COSTOS 47
4.4. Prueba de precisión 48
CONCLUSIONES 51
RECOMENDACIONES 53
BIBLIOGRAFÍA 54

ABREVIATURAS

ACP Accelerator Coherency Port

ADC Analog to digital converter

ASIC Application specific Integrated circuit

CADS Centro de Aguas de la ESPOL

CCU Cache Coherency Unit

CLBs Configurable logic blocks

CPU Unidad central de procesamiento

CVR Centro de Visión y Robótica

DAC Digital to analog converter

DDR Double Data Rate

DSP Digital Signal Processing

ECC Error Correction Code

ESPOL Escuela Superior Politécnica del Litoral

FPGA Field Programmable gate array

GPIO General Purpose Input Output

HDL Hardware Description Language

HMI Human Machine Interface

HPS Hard Processor System

I/O Input/Output

I2C Inter integrated circuits

JTAG Join Test Action Group

LE Logic Elements

MPU Microprocessor Unit Subsystem

NAND Not AND

NO2 Nitrito

OS Operative System

PCB Printed Circuit Board

PIO Parallel Input Output

PLC Controlador lógico Programable

PS Power Source

PWM Pulse Width Modulation

RS232 Recommended Standard 232

RTL Register Transfer Level

SD/MMC Secure Digital/ Multimedia Card Controller

SDRAM Synchronous Dynamic Random-Access Memory

SoC System on a Chip

SoCEDS System on Chip Embedded Development Suite

SOPC System on Programmable Chip

SPI Serial Peripheral Interface

TLB Translation lookaside buffer

UART Universal Asynchronous Receiver-Transmitter

VHDL VHSIC Hardware Description Language

VHSIC Very High Speed Integrated Circuit

SIMBOLOGÍA

A	Amperio
AC	Analog Current
Cx	Capacitor x
DC	Direct Current
Dx	Diode x
GB	GigaBytes
mA	Miliamperio
min	minutos
mL	mililitro
mV	Milivoltio
mW	miliWatts
nm	nanometro
NO2	nitrito
P	Potencia
Rx	Resistencia x
Uf	microFaradio
V	Voltio
Vin	Voltaje de entrada
Vout	Voltaje de Salida
W	Watt
β	Direct Current Gain

ÍNDICE DE FIGURAS

Figura 2.1.- PLCs Compactos Festo FEC FC660, Siemens Logo y S7-200 13
Figura 2.2.- PLC XBM-H Modular 13
Figura 2.3.- PLC's de montaje en rack: Siemens S7-400 PLC y Festo CPX PLC 14
Figura 2.4.- OPLC Unitronics M-90 15
Figura 2.5.- Módulo de expansión MicroLogix 1762 15
Figura 2.6 Bomba Peristáltica Gikfun 12V DC 16
Figura 2.7.- Proceso de captación del líquido de la bomba peristáltica 16
Figura 2.8.- Espectrofotómetro UV-Visible DR 3900 17
Figura 2.9.- Colorímetro CR-410 17
Figura 2.10.- Electroválvula AOMAG 1/4inch DC 12V 18
Figura 2.11 Logo de inicialización del software 19
Figura 2.12 Ventana principal del software Quartus Prime 19
Figura 3.1 Diagrama de flujo del proceso de diseño 20
Figura 3.2.- Diagrama de bloques de la arquitectura del sistema 21
Figura 3.3 Sensor de nivel Taidacent NPN 22
Figura 3.4 Diagrama de flujo del proceso 24
Figura 3.5 Integrado LM317 - Regulador de Voltaje 25
Figura 3.6.- Aplicación estándar para el LM317 25
Figura 3.7 Esquema de fuente regulada de 12V a 3V y 5V 26
Figura 3.8.- Diagrama de pines - Regulador LM7805 26
Figura 3.9.- Transistor TIP31 27
Figura 3.10 Esquema electrónico para el control de un agitador 28
Figura 3.11 Esquema electrónico de conexión de la Electroválvula 29
Figura 3.12 Esquema electrónico para el control de una Bomba peristáltica 29
Figura 3.13 Esquema completo para el manejo de periféricos en EAGLE 31
Figura 3.14 Diagrama de Bloques del sistema controlador. 32
Figura 3.15 Diagrama de entradas y salidas del bloque generador de PWM. Izquierda entradas y Derecha salidas 34
Figura 3.16 Resumen del Sistema Qsys del controlador 34
Figura 3.17 Diferencias entre Hard Processors usados en SOC FPGA's [15] 35

Figura 3.18 Recursos utilizados por el sistema digital diseñado 37
Figura 3.19 Compilación del Sistema Digital para la tarjeta de desarrollo DE0-nano SOC 38
Figura 3.20 Mapa de registros de un PIO (Parallel Input/Output)..... 40
Figura 3.21 Linealización de los datos para obtener una ecuación que los represente 41
Figura 4.1 Diseño electrónico de placa en PCB utilizando Eagle..... 44
Figura 4.2 Resultado final del montaje del circuito electrónico en PCB 44
Figura 4.3 Montaje de la placa junto a la FPGA vista superior..... 45
Figura 4.4 Montaje de la placa junto a la FPGA vista lateral..... 46
Figura 4.5 Salida de monitoreo del programa para el control del dispositivo de medición de nitritos..... 47
Figura 4.6 Mediciones de soluto 1 y 2 tomando como referencia 1mL con FPGA 49
Figura 4.7 Mediciones de soluto 1 y 2 tomando como referencia 1mL con PLC 50

ÍNDICE DE TABLAS

Tabla 3.1 Comparación de transistores 2n222 y TIP31 28
Tabla 3.2 Comparación entre FPGAs de bajo precio 38
Tabla 3.3 Configuración de puerta serial de la FPGA 40
Tabla 4.1 Gasto total de controlador para dispositivo de medición con FPGA 48
Tabla 4.2 Gasto total de controlador para dispositivo de medición con PLC 48

CAPÍTULO 1

1. INTRODUCCIÓN

1.1 Descripción del Problema

Desde la llegada de los PLC al mundo de la ingeniería estos han sido usados como una parte principal en la automatización de procesos ya sean industriales o de algún dispositivo que lo requiera [1]. La ventaja de éstos, es que son especializados para actividades de control; por lo que, todas las rutinas y protocolos de comunicación están desarrollados con el fin de cumplir dicha actividad [2]. Aunque también presentan algunas restricciones que dependen del fabricante, modelo y precio; por lo que, el PLC adecuado se elige de acuerdo con el tipo de proceso que se vaya a realizar; algunas de estas restricciones son el número o tipo de entradas, salidas, protocolos y capacidad de memoria para el programa. Además, los PLCs tienen licencias propietarias para el uso de software para la programación del PLC [3].

El PLC como dispositivo de control está siendo usado en el desarrollo de un dispositivo experimental para medir la concentración de nitritos presente en una muestra de agua. El desarrollo de este dispositivo está a cargo del Centro de Agua de la ESPOL (CADS) en conjunto con el Centro de Visión y Robótica (CVR). Dicho PLC controla la recolección de una muestra de agua desde el río mediante bombas peristálticas; luego, mezcla el agua con reactivos químicos dentro del dispositivo, y, finalmente un colorímetro, el cual no es controlado por el PLC, mide el nivel de luz que la muestra absorbe o refleja.

Con la finalidad de mejorar el producto, los Centros de Investigación mencionados anteriormente contactaron a ECRobotics, una empresa cuyo objetivo es crear sistemas de procesamiento de datos in-situ que puedan ejecutar algoritmos complejos basados en inteligencia artificial. Alinearse con este objetivo y mejorar el dispositivo son cosas difíciles de lograr con el PLC como sistema controlador, ya que, posee un lenguaje de programación de bajo nivel, como diagramas de escalera o de bloques, y el uso de estos depende del tipo de aplicación que se le dará, y, siendo de bajo nivel su uso se limita a realizar las tareas básicas, por lo que, se vuelve complejo el desarrollo de algoritmos

complejos y adaptativos que permitan tener una mayor versatilidad al momento de dar una solución a un problema. El PLC también se ve limitado al poseer diferentes entornos de programación dependiendo de la marca que se esté usando y el costo elevado de las licencias que se tienen que pagar anualmente por el servicio que brindan. En la práctica, un sistema basado en PLC puede requerir de equipos adicionales para poder tener capacidades de adquisición, procesamiento y transmisión de datos, lo cual incrementa su costo aún más.

Otra problemática del PLC es el uso de protocolos de comunicación entre dispositivos, que pueden o no ser inalámbricos, ya que muchos de estos protocolos o dispositivos que permiten establecer comunicaciones son propietarios, es decir, no son interoperables para diferentes marcas limitando el crecimiento y escalabilidad de algún sistema. En cambio, la FPGA puede usar una amplia gama de sensores, actuadores, y antenas que están disponibles a un menor costo y que pueden ser integradas aún si son de diferentes marcas, esto permite implementar comunicaciones inalámbricas fácilmente, ya que, en algunos casos se cuenta con soporte del fabricante y los aportes de la comunidad [3].

1.2 Justificación del Problema

1.2.1 Justificación Técnica

Los PLC son sistemas basados en controladores que pueden ser usado para activar sensores, actuadores y ejecutar rutinas de control. Los PLCs modernos pueden realizar tareas de controles de forma simultánea y pueden ejecutar sus rutinas a altas frecuencias, pero no pueden realizar algoritmos complejos y adaptativos [4]. En cambio, el FPGA al ser una tarjeta lógica programable puede implementar de forma programada diferentes componentes de hardware que procesan señales digitales o análogicas, y que a diferencia del PLC es muy flexible; dado que, permite crear diferentes dispositivos como CPU, controladores, interfaces en una sola tarjeta.

Un FPGA con un sistema digital y un CPU que ejecuta un sistema operativo es conocido como un “System on a Chip” (SoC), refiriéndose a que en una sola tarjeta de desarrollo se encuentran integrados gran parte de los elementos que componen un computador. El SoC que ECRobotics propone puede ser usado

para realizar procesamiento inteligente de datos, y, presenta varias ventajas para el desarrollo de este proyecto.

Primero, Al tener un sistema operativo integrado, existe la facilidad de usar un lenguaje de programación de alto nivel, el cual, puede controlar elementos de hardware con tiempos de respuestas pequeños. Además, permite ejecutar varias tareas de forma paralela, ya que, el sistema operativo dentro del FPGA es multitarea. Así, el control o lectura de los elementos que componen el sistema de medición puede ser realizado desde algún programa que funcione como un proceso dentro del sistema operativo, y, a su vez, en una tarea paralela se puede analizar los datos que el colorímetro envía a través de una interfaz de programación (API) que el fabricante del dispositivo provee.

Segundo, se pueden combinar diferentes tipos de sensores, ya que el FPGA puede actualizar los componentes de control o captura de datos si el proceso de preparación de la muestra cambia. Por ejemplo, si se requieren mezclar más reactivos, se deberán agregar más actuadores o sensores, lo que implica el uso de una mayor cantidad de entradas/salidas del controlador, lo cual no será un problema para el FPGA, puesto que, este cuenta con una gran cantidad de GPIO (Pins de entrada y salida de propósito general) y varias interfaces de comunicación como I2C, SPI, entre otras. En cambio, para el PCL se deberá adquirir módulos de expansión de entradas/salidas e incluso en algunos casos adquirir un controlador diferente, no será necesario adquirir otro equipo de control, a diferencia de usar PLCs, lo cual si ocurre. Simplemente se pueden agregar más bloques de control o cambiar su programación en el FPGA [5]. Además, con la investigación necesaria se puede agregar sensores diferentes que permitan mejorar el proceso de estimación de la concentración de nitritos e incluso que permitan derivar otros parámetros de interés de manera indirecta.

Finalmente, al ser un dispositivo programable el uso de FPGA D0 Nano reduce los tiempos de adaptación de nuevas tecnologías y su precio de mercado ronda los 130USD para la escala del problema que se espera resolver [6].

1.2.2 Justificación Económica

Hoy en día el mercado ofrece una amplia gama de dispositivos de control que se adaptan a las necesidades del usuario. Entre ellas tenemos el PLC

micrologix 1100 con un valor en el mercado de $348.90 y una FPGA D0 Nano SoC a $130 dólares, evidentemente la diferencia de precios es notable. Además, el PLC nombrado tiene otro punto en contra, la falta de entradas o salidas digitales, lo cual puede solucionarse con un módulo de expansión, pero, el costo final del controlador incrementa considerablemente, esto lo diferencia del FPGA ya que este cuenta con aproximadamente 80 entradas/salidas sin necesidad de módulos adicionales [5].

1.2.3 Justificación Financiera

El proyecto cuenta con el respaldo del cliente ECRobotics, propietario del dispositivo experimental, ya que brinda las herramientas y materiales necesarios para el diseño del prototipo de sistema controlador basado en FPGA. El beneficio que obtiene nuestro cliente es poder extender la funcionalidad de su plataforma sin tener que comunicarse con un PLC externo. Esto se traduce en menores costos de producción y un dispositivo de calidad capaz de cumplir con los requerimientos del cliente. Además, que con el reacondicionamiento del dispositivo experimental se puede incrementar la productividad y eficiencia de la maquinaria que maneja en su interior.

1.3 Objetivos

1.3.1 Objetivo general

- Diseñar y construir un prototipo basado en FPGA y sistemas embebidos, capaz de controlar dispositivo de medición e identificación de niveles de NO2 en cuerpos de agua.

1.3.2 Objetivos Específicos

- Conocer la arquitectura, funcionamiento y parámetros necesarios que el dispositivo de medición de NO2 usa actualmente, para establecer los requerimientos del nuevo controlador.
- Realizar un diagrama de flujo con el proceso completo que el dispositivo experimental de medición de nitritos realiza, para la identificación de nitritos; como ayuda para la realización del programa.

- Diseñar interfaces electrónicas para la interconexión de los elementos de potencia con las entradas/salidas digitales o analógicas de la FPGA.
- Diseñar componentes de software que sirvan como drivers para la interconexión de los elementos eléctricos o electrónicos, que usa el dispositivo de medición de NO2.
- Implementar el sistema controlador para evaluar los resultados y realizar ajustes de calibración que permitan su correcta operación.

1.4 Estado del Arte

Basado en los trabajos previos realizados en el mismo campo, pero con aplicaciones diferentes, podemos conocer información útil previa a la realización del proyecto.

Los diferentes estudios se basan en si en la implementación de sistemas en FPGA orientadas a diferentes requerimientos los cuales se van a presentar en orden cronológico, esto permite conocer las mejoras realizadas a las tecnologías existentes que se han ido desarrollando en el tiempo. Por lo cual, a través de este análisis se puede identificar el precedente de lo que se espera lograr en el proyecto presente y futuros similares.

La necesidad de tener un controlador con más capacidad de procesamiento ha llevado a la industria del moldeo por inyección a buscar alternativas que puedan cumplir con la demanda de todos los procesos que se requieren para llevar a cabo la realización del producto final, como lo propone el siguiente artículo “Controlador modular y reconfigurable para máquina de inyección de plástico basado en FPGA”. El moldeo por inyección de plástico es un proceso que maneja variables como: velocidad, presión, temperatura, posición y otros eventos discretos que requieren de una capacidad de cómputo muy superior a la que ofrece un PLC, el cual tiene una limitante en su arquitectura, condicionando el desempeño del proceso. La solución fue usar un dispositivo dispositivo de alto desempeño como la FPGA, en la cual se diseñó un sistema modular que maneje todos los procesos involucrados; y, se añadió un módulo de comunicación. Las pruebas del sistema se hicieron en una máquina de inyección Husky, dando como resultado un sistema que controle el sistema de inyección de forma eficiente, lo que convierte al FPGA en una opción viable para empresas

pequeñas y medianas que necesitan incrementar el nivel de productividad de su maquinaria [7].

En el siguiente proyecto “Implementación de un FPGA para control y adquisición de datos mediante una unidad de telemetría remota” se decidió como parte de un proyecto de renovación de equipos, usar como controlador un FPGA con la finalidad de mitigar la carga de procesamiento de datos y la necesidad de adicionar un sistema de comunicación. Con esto lograron obtener un sistema de chip programable que satisfaga todas sus necesidades y con la capacidad de conectarse a internet. Como resultado se obtuvo un dispositivo compacto, bajo costo y de alta velocidad de procesamiento [2] .

Además “Implementación en dispositivos FPGA de controladores para sistemas lineales inestables con retardo de tiempo”, el cual usa el FPGA para compensar los retardos y lograr más estabilidad en el sistema, cuando este actúa en lazo cerrado. El resultado obtenido fue un “control” estable gracias a la predicción de la salida del sistema, y, gracias a la capacidad de procesamiento de la FPGA, se puede incluir un modelo “Real” para la realización de pruebas en tiempo real, sin necesidad de contar con el sistema físico sino más bien emulado[8].

Al revisar los trabajos relacionados se nota que básicamente se ha usado al FPGA como tarjeta de desarrollo y controlador de dispositivos digitales que no manejan potencia, y en uno que otro caso tratan de emular un PLC, nuestro proyecto se diferencia, ya que no solo se reemplazará al PLC, sino, que se harán las interfaces necesarias para que este maneje los elementos de potencia y además combinar el sistema embebido de la FPGA para desarrollar algoritmos más complejos que realice de forma automática el análisis de los datos obtenidos del sistema de medición.

1.5 Alcance

El proyecto incluye la realización de las interfaces electrónicas de conexión entre los dispositivos de potencia (motores, bombas peristálticas, etc.) usados dentro del sensor experimental y la FPGA.

Se incluye también la realización de los módulos o programas en la FPGA para el control de los dispositivos mencionados anteriormente incluyendo sus

protocolos de comunicación y el programa completo que maneje el sensor experimental y todas las etapas necesarias para obtener las mediciones.

Incluye la realización de un algoritmo de análisis de los datos obtenidos, manejado desde el Sistema Operativo de la FPGA y además una salida serial para el monitoreo del proceso de medición.

Este proyecto no incluye la integración con la red SCADA, ya que se pretende únicamente darle más capacidades al dispositivo de medición y hacerlo más “inteligente”, de tal forma que pueda ser usado en futuro por la plataforma de ECRobotics, la empresa contratante

1.6 Metodología

El proyecto se dividirá en varias etapas, de las cuales, la etapa inicial será conocer el funcionamiento de cada uno de los elementos que conforman el sistema de medición del dispositivo experimental. En esta etapa se deberá conocer cómo dichos elementos se involucran en el proceso de medición de la concentración de nitritos, para luego poder conocer correctamente el proceso que el dispositivo realiza.

Con el conocimiento de los elementos que conforman al sistema, se estudia el proceso que el PLC realiza para las mediciones, con el objetivo de obtener un diagrama de flujo que describa dicho proceso, lo que luego facilitará la implementación del mismo en un lenguaje de programación de alto nivel dentro del sistema embebido de la FPGA.

De acuerdo con el diagrama de flujo se realizará una estimación de la capacidad de procesamiento necesaria para realizar todo el proceso, y con esta se definirá el FPGA requerido.

Luego se realizarán las adecuaciones eléctricas y/o electrónicas necesarias para cada sensor o actuador, de tal forma que estos puedan ser manejados por los niveles de voltaje y corriente con los que opera el FPGA.

Conociendo el cómo se conectarán los elementos de control a la FPGA se deberá implementar un circuito digital programado haciendo uso del VHDL que

servirá como interfaz (Driver) entre estos elementos y el sistema operativo instalado en la FPGA.

Con el diseño de control completo y su implementación se construirá un algoritmo capaz de analizar los resultados y dar conclusiones acerca de los datos obtenidos.

CAPÍTULO 2

2. MARCO TEÓRICO

2.1 Tarjeta lógica programable FPGA

Por sus siglas en ingles es Field Programmable Gate Array o en español Arreglo de Compuertas Programables en Campo. La FPGA es una tarjeta lógica programable que posee subcircuitos digitales programables y configurados para una amplia gama de aplicaciones, es decir, que internamente se configura para que sea un dispositivo con una tarea específica y que luego puede ser reconfigurado. La capacidad de ser reprogramable lo hace un dispositivo con capacidades ilimitadas, permitiendo realizar diseños digitales eficientes y a la medida del proyecto que se desea realizar [9].

El FPGA puede ser programado mediante dos tipos de lenguaje populares como lo son VHDL y Verilog. El tipo de programación usado no es el tradicional en donde se introducen instrucciones, sino que se “programa” la descripción del hardware, es decir se realiza el diseño de la arquitectura a nivel de hardware [10].

2.2 Características de una FPGA

2.2.1 Arquitectura Programable

Poseer una arquitectura programable significa que podemos optimizar los procesos que realizara la tarjeta y realizar cambios para mejorar el rendimiento. El lenguaje de programación de bajo nivel como VHDL describirá la forma en cómo será usado el hardware [11].

2.2.2 Desarrollo acelerado

Los FPGA en su interior posee diminutos semiconductores basados en matriz de bloques configurables o CLBs, es decir trabajan a nivel lógico. Esto permite que se puedan traer versiones con mayor capacidad de procesamiento en menos tiempo que la competencia, ya que requiere de ciclos de fabricación más extensos [11].

2.2.3 Bajo costo de operación

Los dispositivos ASIC que son circuitos integrados dedicados a aplicaciones específicas, pueden llegar a ser más económicos que una FPGA, pero poseen

un hardware estático y no se pueden realizar cambios a futuro, esto lo diferencia de una FPGA, ya que, este puede realizar mejoras en su arquitectura fácilmente, cambiando simplemente la programación de su descripción de hardware. Un ejemplo muy claro es cuando tenemos un dispositivo ASIC funcionando y se requiere de una actualización del sistema, esto hace que el ASIC deba ser reemplazado por uno más actualizado o mejor que maneje las nuevas configuraciones; pero, con un FPGA no hay necesidad de cambiar el dispositivo, sino realizar la reconfiguración de su hardware de tal forma que se acople a las nuevas necesidades del sistema [11].

2.2.4 Integración de hardware

Los FPGA poseen sistemas de entrada y salida, procesadores y muchas funciones más. Esto significa que requiere de menos dispositivos externos para llevar a cabo más funciones. A diferencia de los dispositivos ASIC que tienen en su interior chips conectados en paralelo, es decir si uno de ellos falla significa que todo el dispositivo fallara [11].

2.3 Aplicaciones

Debido a la versatilidad y flexibilidad del FPGA, el abanico de aplicaciones es muy amplio. Su principal aplicación está orientada al procesamiento digital de señales (DSP), circuitos digitales, procesamiento de datos y comunicaciones [12].

2.3.1 Sistema de inteligencia artificial

Hoy en día cada vez son más los dispositivos que poseen un sistema de visión artificial como robots, cámaras de vigilancia, celulares, etc. Estos dispositivos necesitan reconocer objetos, reconocer rostros e incluso conocer distancias para conocer su posición e interactuar mejor con el entorno. El análisis de un alto volúmenes de imágenes requiere de una capacidad de procesamiento elevado y además deben ser procesadas en tiempo real [12].

2.3.2 Radio definido por software

Un radio está compuesto por una antena que se encarga de enviar y recibir la señal y un hardware que procesa la señal. Este hardware era dedicado y

modular, es decir, existía un hardware específico para cada función de un sistema de telecomunicaciones. Hoy en día ya no se usan chips dedicados a aplicaciones específicas, sino más bien se usa un procesador de propósitos generales y dentro de este se programan todos los componentes necesarios para el sistema de comunicaciones, esto es conocido como SDR (Radio Definido por Software). Algunos FPGA cuentan con sistemas ADC y DAC integrados, y, en conjunto con su capacidad de reconfiguración de hardware, los hace un dispositivo perfecto para un sistema de Radio Definido por Software [12].

2.3.3 Seguridad

El envió de información de forma segura es primordial, ya que, realizar procedimientos como una compra online o realizar una transferencia bancaria sin una seguridad adecuada en los datos transferidos, puede suponer una gran pérdida de recursos, por esto, la capacidad de un FPGA de manejar un gran volumen de datos y bloques optimizados para realizar opciones aritméticas que permiten la encriptación eficiente al enviar información es una gran opción para aplicaciones de seguridad informática [12].

2.4 Controlador Lógico Programable PLC

Un PLC o Programmable Logic Sistem por sus siglas en ingles. Es un dispositivo dedicado a realizar procesos electromecánicos en el área de automatización industrial. Procesos como el control de maquinaria en líneas de montaje hasta el control de iluminarias son unas de las muchas aplicaciones que son capaces de realizar [13].

Los PLC son capaces de llevar a cabo varios procesos a la vez, entre sus características tenemos que poseen múltiples entradas y salidas, resistencia a la vibración, rango alto de temperatura, inmunidad al ruido eléctrico y resistentes a impactos. Ideal para procesos con condiciones ambientales que requieran de un dispositivo resistente [13].

El PLC es un dispositivo que tiene la capacidad de conexión directa, es decir que puede ser conectado a valores de tensión y corriente de niveles industriales.

2.4.1 Ventajas y desventajas del uso de un PLC

Ventajas

- Facilidad de programación por no especialistas.
- Capaz de comunicarse con otros PLCs y ordenadores.
- Ideal para uso industrial ya que soporta humedad, polvo, vibraciones, temperatura y ruido.
- Seguridad de control de procesos.
- Manejo de múltiples actuadores.
- Rápida detección de averías.
- Bajo costo de instalación, mantenimiento y operación.

Desventajas

- El costo inicial para autorizar un proceso es elevado, no es recomendado implementarse en tareas sencillas.
- Mano de obra con personal especializado.
- Para automatizar una tarea es necesario tener en cuenta todos los detalles al momento de programar.

2.4.2 TIPOS DE PLC

PLC COMPACTOS

Los PLC compactos traen todo en un solo paquete, incluyen PS, CPU, módulo de entrada y salida de hasta 30 slots fijo, canales de comunicación para programar el PLC e interfaz hombre maquina (HMI). Algunos modelos incorporan una conexión de entrada de alta velocidad al controlador, dos entradas o salidas analógicas. En caso de tener la necesidad de aumentar el número de entradas o salidas es necesario adquirir un módulo. Normalmente usados como sustitutos de relés [13].

Figura 2.1.- PLCs Compactos Festo FEC FC660, Siemens Logo y S7-200

PLC Modular

Este tipo de PLC viene equipado con más funciones y mejores características en su hardware, que un PLC compacto. La SM, CP, CPU y accesorios adicionales que se pueden añadir por separado. Posee un número limitado de lugares en donde se pueden colocar módulos, pero en caso de llegar a necesitar más pueden aumentarse. Los PLC modulares integran una mayor cantidad de entradas y salidas, mayor capacidad de memoria maneja programas más robustos, lleva a cabo múltiples tareas. Usualmente se emplean para el procesamiento de datos, posicionamiento, monitoreo, comunicación y servicios web [13].

Figura 2.2.- PLC XBM-H Modular

PLC de montaje en RACK

Este tipo de PLC tienen la misma capacidad que un PLC modular, pero su principal diferencia está en el rack y bus de datos donde los módulos son colocados. Este PLC tiene ranuras para colocar módulos y un bus integrado para la comunicación entre módulos. Su principal ventaja es la capacidad de realizar un intercambio rápido de información de datos entre módulos lo que se traduce en un menor tiempo de respuesta a diferencia de otros módulos que solo disponen de un panel frontal con interfaz HIM [13].

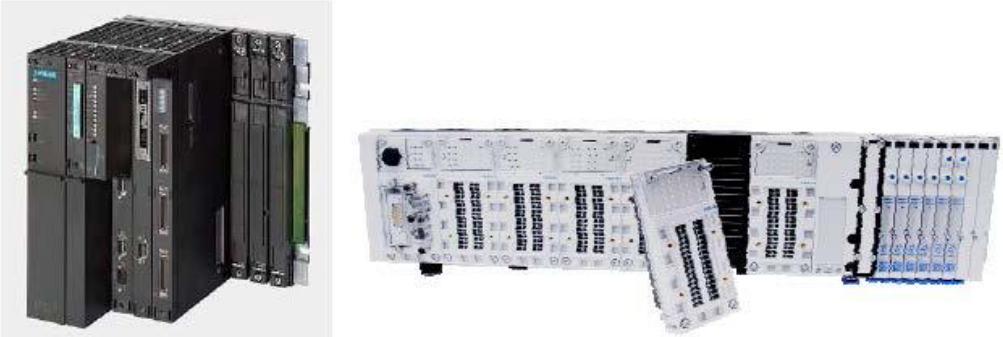

Figura 2.3.- PLC's de montaje en rack: Siemens S7-400 PLC y Festo CPX PLC

PLC con panel operador

Estos PLC incluyen una interfaz HIM que facilitan el monitoreo y control de procesos y maquinas. Una interfaz HIM es una pantalla táctil si es un modelo actual o un monitor y teclado si es un modelo antiguo. Su principal ventaja es que al traer un panel operador aparte no es necesario programarlo por separado, sino que toda la programación se realiza por medio de una herramienta o software [13].

Figura 2.4.- OPLC Unitronics M-90

2.5. Módulo de expansión de E/S digitales MicroLogix 1100

Mediante este módulo se pueden ampliar el número de E/S que serán necesarias para su proyecto. Este es un diseño modular sin rack que es más económico y reduce los costos por piezas de repuesto en caso de desperfecto.

- Modelo MicroLogix 1762
- Manejo de voltaje alterno (AC) y continuo (DC)
- Relé de DC/AC
- Disponible en 8, 16 o 32 E/S digitales
- Modulo combinado con 6 salidas de contacto y 8 entradas

Figura 2.5.- Módulo de expansión MicroLogix 1762

2.6. Bomba peristáltica

Una bomba peristáltica también conocida como bomba de desplazamiento positivo porque tiene una parte por donde succiona el fluido y otra por donde es expulsado. El líquido es transportado por un tubo flexible que está dentro de la cubierta circular, además tiene un rotor con unos rodillos los cuales al girar van a empujar el liquito por el tubo flexible.

Figura 2.6 Bomba Peristáltica Gikfun 12V DC

Funcionamiento

Los rodillos al interior de la cavidad circular al girar comprimen el tubo flexible que crea un vacío que succiona al fluido a través del tubo flexible. La ventaja de este tipo de bombas es que al ser comprimida la manguera se genera un flujo positivo del fluido evitando que se generen reflujos, de esta manera no es necesario el uso de valvulares reguladoras.

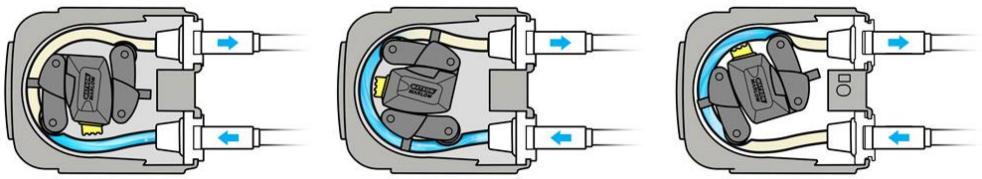

Figura 2.7.- Proceso de captación del líquido de la bomba peristáltica

2.7. Espectrofotómetro

Es un instrumento de análisis óptico que permite comparar el nivel de radiación absorbida por una muestra de solución que contiene una cantidad desconocida de soluto y una que tiene una cantidad conocida de ese soluto.

El espectrofotómetro proyecta un haz de luz monocromático sobre la muestra y mide la cantidad de luz absorbida sobre ella. Como resultado tenemos:

- Información de la naturaleza de la sustancia que posee la muestra
- Cantidad de sustancia presente en la muestra.

Figura 2.8.- Espectrofotómetro UV-Visible DR 3900

2.8. Colorímetro

Es una herramienta que mide la absorbancia de una sustancia, ésta consiste en que la absorbancia es directamente proporcional a la concentración de la sustancia en estudio. El colorímetro mide la concentración de una sustancia a partir de la identificación de colores y sus diferentes variantes por ello si una sustancia tiene mayor concentración presentara mayor absorbancia.

Cada sustancia es capaz de absorber diferentes frecuencias que se traducen en colores. Para realizar las mediciones se usa un filtro que seleccionará el color de luz que será absorbido por el soluto, esto permite tener un mejor resultado en la medición de la muestra. Nótese que el color que más absorbe el soluto es el opuesto al color que presenta la muestra.

Figura 2.9.- Colorímetro CR-410

2.9. Electroválvula

Una electroválvula funciona con un mecanismo electromecánico y está diseñada para permitir o impedir el paso de fluido. El mecanismo es accionado a través de una bobina

cuyo voltaje de accionamiento dependerá de las especificaciones del fabricante. Por lo general posee únicamente dos estados, abierto o cerrado y pueden ser usadas en un sin número de situaciones para el control de fluidos.

Figura 2.10.- Electroválvula AOMAG 1/4inch DC 12V

2.10. Intel Quartus Altera

Es un programa de diseño para dispositivos lógicos programables que permite al usuario la síntesis de diseños HDL, análisis de diagramas RTL, realizar simulaciones de un diseño, configurar un dispositivo de acuerdo con el objetivo al que desea llegar el programador. Quartus posee un apartado prime que permite al usuario usar VHDL para la programación del hardware. Además, se puede realizar la edición de circuitos lógicos en una interfaz gráfica [14].

Características de Intel Quartus Prime:

- Incluye SOPC Builder, herramienta que elimina los procesos manuales al momento de generar la interconexión de una tarjeta lógica y verificar su funcionamiento mediante un banco de pruebas.
- DSP Builder, herramienta que permite a Quartus tener una conexión directa con Matlab y simulink.
- SoCEDS, herramienta que contiene ejemplos de aplicaciones que ayudan al desarrollador a crear nuevas herramientas.
- Qsys herramienta que permite la integración de sistemas, es una versión más avanzada que SOPC Builder.

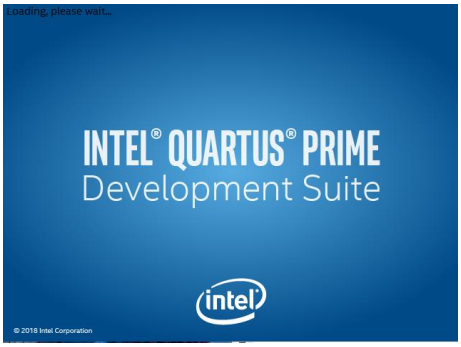

Figura 2.11 Logo de inicialización del software

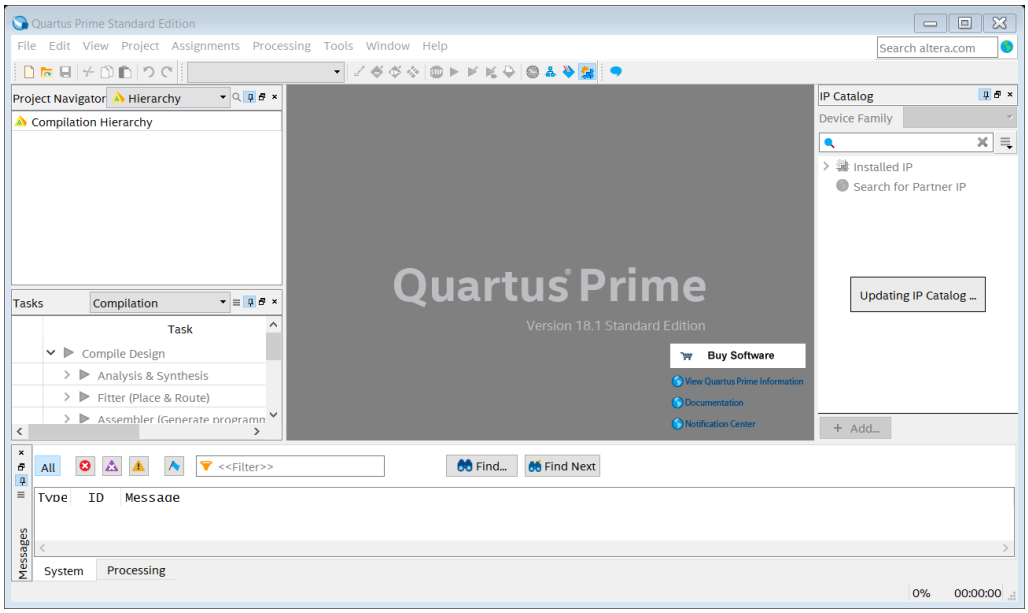

Figura 2.12 Ventana principal del software Quartus Prime

CAPÍTULO 3

3. DISEÑO DE LA SOLUCIÓN

El proceso de diseño consiste en conocer la arquitectura, funcionamiento y características eléctricas del dispositivo de medición. Una vez que se tenga una idea detallada del proceso y como lo lleva a cabo el aparato. Se empezará por la elaboración de un diagrama de flujo del proceso de medición de nitritos, elección de los componentes necesarios para llevar a cabo el reemplazo del controlador. Seguido del diseño de las interfaces electrónicas que permitirán que el FPGA pueda manejar cada uno de los periféricos del dispositivo de medición sin ningún problema. También se lleva a cabo el diseño del software que permita la interacción y control de cada uno de los periféricos para luego realizar el montaje del FPGA y sus adaptaciones. Para finalizar se realizará pruebas de funcionamiento y calibración.

A continuación, en la Figura 3.1 se presenta un diagrama de flujo que detalla el proceso de diseño del sistema controlador basado en FPGA.

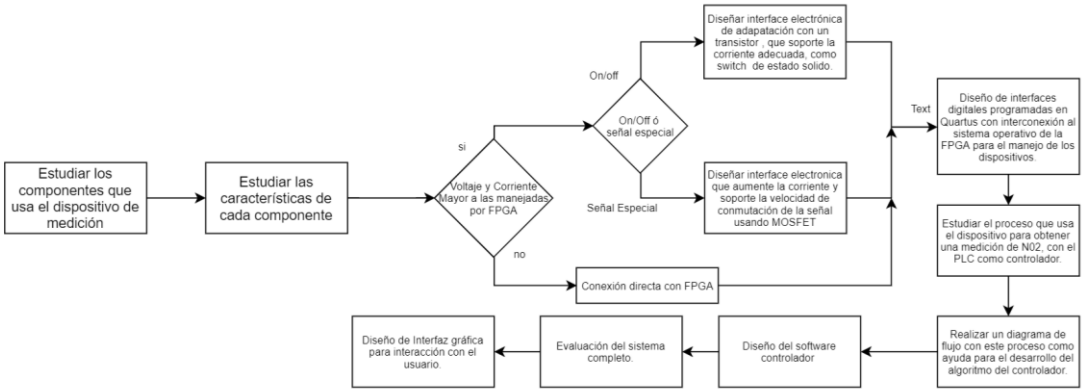

Figura 3.1 Diagrama de flujo del proceso de diseño

3.1. Funcionamiento del dispositivo de medición de NO_2

En este apartado se realiza el estudio del funcionamiento previo del dispositivo experimental de medición de NO_2. Este dispositivo basa su funcionamiento en la colorimetría, es decir, internamente realiza reacciones químicas de ciertos compuestos con la muestra de agua; para que al final esta se torne de un color específico que represente la cantidad de nitrito presente en la muestra. La obtención de datos de esta mezcla “coloreada” se realiza a través de

espectrofotometría que analiza en base a longitudes de ondas el color de la muestra, y, con esto determina la concentración de $N0_2$ presente en la muestra.

3.1.1 Arquitectura

El dispositivo de medición posee un PLC que es el encargado de recoger la señales de cada uno de los sensores y accionar los actuadores en cada una de las etapas del proceso de medición de nitrito. El PLC cuenta con una conexión a un servidor que le permite guardar cada uno de los datos obtenidos, además cuenta con un ordenador enlazado al servidor que tiene una aplicación para el monitoreo de las mediciones hechas por el dispositivo. El esquema de la arquitectura de funcionamiento del dispositivo se muestra en la Figura 3.2 a continuación.

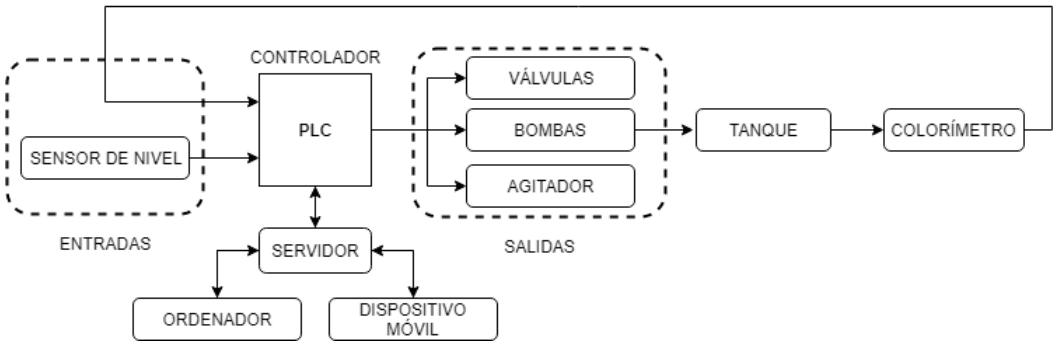

Figura 3.2.- Diagrama de bloques de la arquitectura del sistema

3.1.2 Componentes del sistema

PLC Micrologix 1100 funciona como controlador del sistema, cuenta con 10 entradas digitales, 6 salidas tipo relé y comunicación Ethernet. Se le puede añadir un módulo de expansión de 16 salidas digitales (1762 IQ16) y un módulo de 16 salidas tipo relé (1762 OW16).

Sensor de nivel IS NPN opera con un nivel de voltaje de entre 5 y 12V. Este sensor no necesita estar en contacto con el líquido para proporcionar las medidas. Es un sensor digital, es decir, se encarga de indicar si existe o no liquido presente en el nivel o altura del recipiente en el que se encuentre colocado. Si el nivel de la solución se encuentra bajo acciona una alarma para que el operador se acerque a llenar los recipientes.

Figura 3.3 Sensor de nivel Taidacent NPN

Electroválvulas son las encargadas de controlar el paso de líquidos a través de las mangueras. Posee dos vías normalmente cerradas con un diámetro externo de 6 mm en sus conectores.

Bombas peristálticas se encarga de desplazar los fluidos con un caudal contante por las mangueras. Las bombas de la marca Gikfun AE1207 manejan un caudal de entre 0 a 100mL/min, posee cañerías con diámetro interno de 3mm.

Mangueras, usa dos tipos de manguera una con un diámetro interno de 3 mm y diámetro externo de 4 mm, la segunda manguera tiene un diámetro interno de 2 mm y diámetro externo de 4 mm. Es importante recalcar que usa la maguera con menor diámetro para tener mayor control de la solución al momento de medirla, es decir si se necesita 0,5 mL de solución se requerirá que el fluido se encuentre en 159,14 mm de manguera con 2 mm de diámetro interno.

Recipientes color ámbar, el dispositivo usa este tipo de recipientes para proteger las soluciones (reactivos) de la presencia de luz, ya que son de características fotosensibles.

3.2. Proceso de medición de nitritos

El proceso se divide en cuatro etapas, reduciendo al máximo la intervención humana y obtener los resultados de concentración de nitritos en la muestra.

Etapa Cero: Calibración. El dispositivo realizar un proceso de calibración que permitirá obtener una representación gráfica de absorbancia y concentración de nitrito. Se mide 1ml de la solución patrón y 100ml de solución desionizada, se obtiene una mezcla con 1 umol/L de concentración. Se toma muestras de $0{,}25\ \ 0.50\ \ 0.75\ \ 1\ y\ 1.25\ mL$, se completa con 25mL. Estas muestras son analizadas por el colorímetro que medirá el nivel de absorbancia de cada muestra. Mediante la técnica de regresión lineal se obtiene la gráfica que obedece a la ecuación 3.1, la cual representara la conversión entre absorbancia y concentración para mediciones futuras, con muestras de niveles de concentración desconodidas.

$$Y = mX + b \tag{3.1}$$

Donde Y representa el valor de absorbancia, X la concentración de la muestra y b el intercepto en el eje X.

Etapa Uno, denominada dosificación, realiza la medición del volumen de muestra y volumen de soluto necesario para cada mezcla con la ayuda de las bombas peristálticas y el control con las electroválvulas.

Etapa Tres, denominada medición, la muestra ya posee todos reactivos y después de entrar en reposo por un determinando tiempo será analizada por el colorímetro.

Etapa Cuatro, denominada limpieza, una vez realizada las mediciones correspondientes se procede a desechar la solución y verter agua destilada en los recipientes usado para mantener los recipientes libres de residuos que puedan contaminar las siguientes muestras y alterar los resultados de medición.

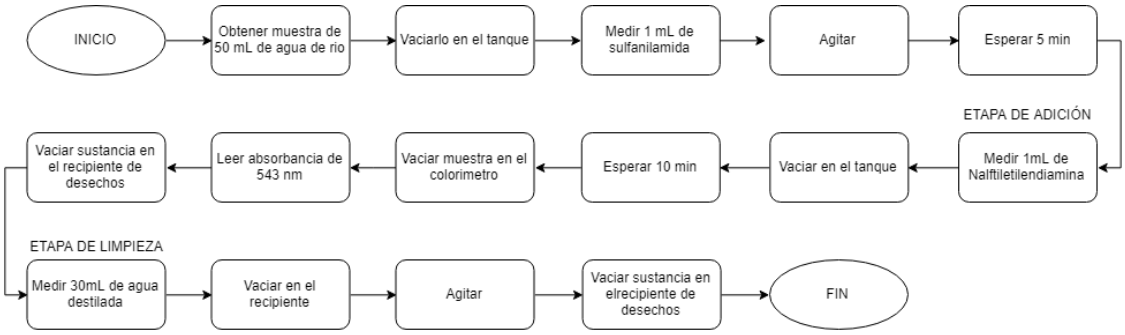

Figura 3.4 Diagrama de flujo del proceso

3.3. Esquema de interfaces electrónicas

La FPGA al ser una tarjeta programable tiene una entrada de alimentación de 5v y es capaz de generar voltaje de salida de 3.3v y 5v en sus salidas digitales. En cambio, los actuadores y elementos que serán usados en el sistema de medición trabajan con un nivel de 12 voltios y una corriente mayor a la que soporta el FPGA, por esa razón es necesario el diseño de una interfaz de acondicionamiento electrónico. El esquema maneja 14 electroválvulas, 5 bombas peristálticas, 1 agitador, 2 fuentes reguladas stepdown que permite manejar voltaje de 3v y 5v.

Cada circuito de control es asignado a cada salida GPIO que posee la FPGA, de esta forma todo estará integrado en un solo circuito que converge en un bus de datos.

Diseño de fuentes reguladas

En la Figura 3.7 se presenta el esquema de dos fuentes step-down que reducen el nivel de tensión. Se decidió usar este tipo de fuente lineal porque es de fácil construcción, usa pocos elementos, eficaz, baratas y se adecúa a las necesidades del proyecto.

El LM317 es un regulador de voltaje ajustable con un rango de 1.25V a 37V en la salida e intensidad de 1.5A. Posee tres pines: ADJ, entrada y salida.

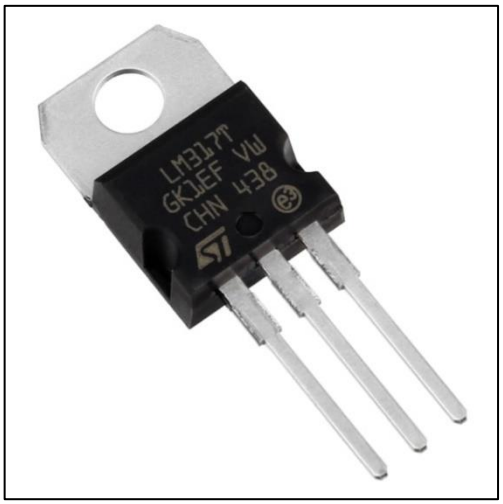

Figura 3.5 Integrado LM317 - Regulador de Voltaje

En el datasheet del LM317 el fabricante presenta un esquema de aplicación típica que ilustra la configuración de una fuente regulada.

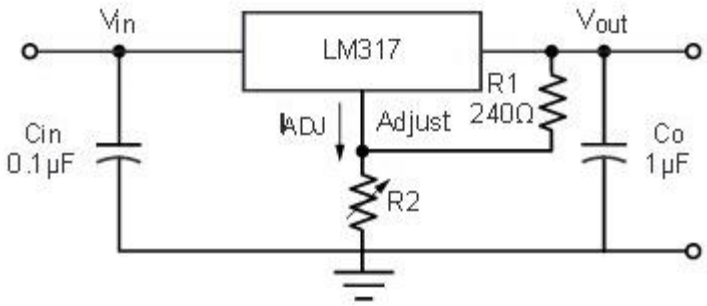

Figura 3.6.- Aplicación estándar para el LM317

El voltaje de salida no depende de los condensadores que se encuentran tanto a la entrada como a la salida como se muestra en la Figura 3.6. El propósito de colocar C_{IN} y C_o es de mejorar la respuesta transitoria y mejorar el rechazo del rizado de la señal, obteniendo como resultado una señal más limpia y lineal posible.

El nivel de tensión de salida del regulador de voltaje de la Figura 3.6 puede ser calculado a partir de la ecuación 3.2, donde V_{Ref} tiene un valor de 1.25V y I_{ADJ} con 50uA. En la ecuación 3.3 se observa el cálculo de las resistencias R2 y R1 para lograr una salida de 3V a la salida.

$$V_o = V_{Ref}\left(1 + \frac{R_2}{R_1}\right) + I_{ADJ}R_2 \qquad (3.2)$$

$$V_o = 1.25\left(1 + \frac{330}{240}\right) + (50uA)(330) = 2.99V \quad (3.3)$$

En la parte superior de la Figura 3.7 se tiene una fuente de 12V a 3V y dada las características mencionadas anteriormente del LM317, en conjunto con la obtención de resistencias comerciales (resistencias de $240\ y\ 330$) la obtención de 3V a la salida de la fuente fue algo sencillo y económico.

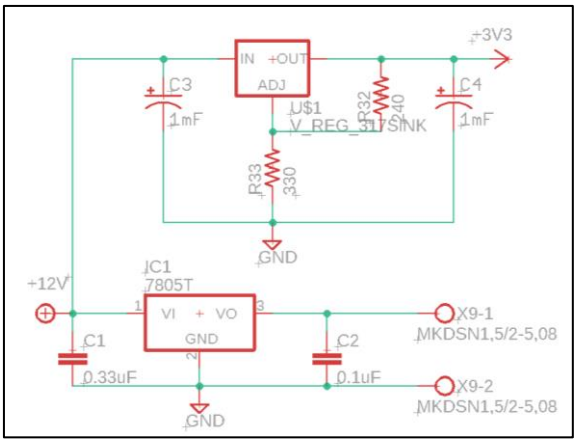

Figura 3.7 Esquema de fuente regulada de 12V a 3V y 5V

En la parte inferior de la Figura 3.7 se tiene una fuente regulada de tipo stepdown que va de 12V a 5V que será usada para la alimentación de la FPGA.

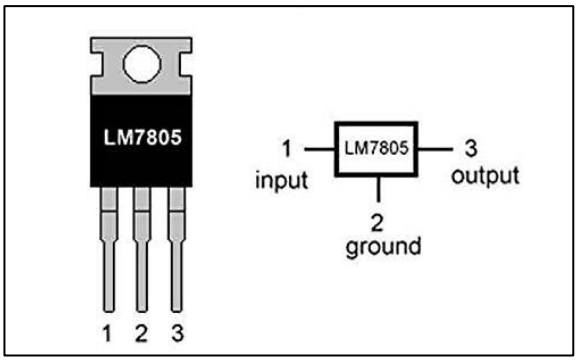

Figura 3.8.- Diagrama de pines - Regulador LM7805

Una solución para obtener un nivel de tensión de 5V es usar un regulador de voltaje LM7805 que posee una limitante en la tensión de entrada de hasta 35V. Otra limitante de este integrado es el calor que genera, si la diferencia de voltaje entre la entrada y la salida es demasiado grande, este no es un problema para el presente diseño, ya que en este se requiere un voltaje de entrada de 12V y un

voltaje de salida de 5V. En caso de tener una diferencia muy grande, se deberá usar un disipador de calor.

Diseño de interfaz electrónica de un agitador

Dentro de los elementos que se necesitan controlar esta un agitador, el cual será el encargado de mezclar los líquidos que llegan al recipiente para formar una mezcla uniforme y coloreada que luego será analizada en el colorímetro. Esta interfaz debe ser capaz de funcionar a un voltaje de 3V y una corriente mínima de 400mA.

La mejor opción de acuerdo con los requerimientos del agitador fue usar un transistor TIP31 Figura 3.9 que es un transistor usado para conmutación en circuitos de potencia, y, es capaz de manejar corrientes de hasta 3A y soportar un voltaje en la juntura base-emisor de 100V. Estas especificaciones se adecuan a la aplicación que se le dará en el presente proyecto.

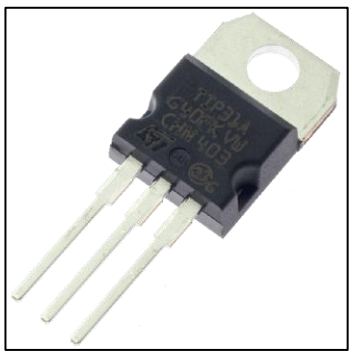

Figura 3.9.- Transistor TIP31

El circuito de la Figura 3.10 es el esquema para el control del agitador, su funcionamiento se basa en un switch de estado sólido, es decir cada vez que llegue la señal de voltaje en la base el transistor TIP31 se polarizara y entrara en zona de saturación permitiendo que el agitador se accione, caso contrario el transistor estará en zona de corte apagando el agitador. En el colector se tiene un diodo que hace de protector de corriente, impidiendo que corrientes inversas puedan dañar el agitador, el circuito o la FPGA.

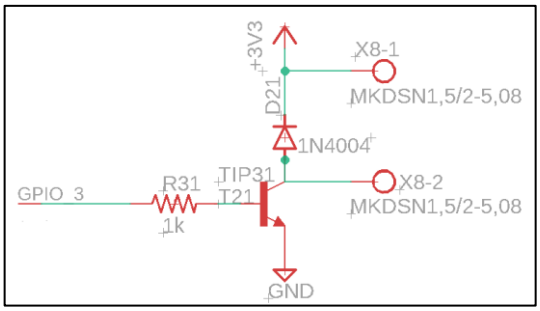

Figura 3.10 Esquema electrónico para el control de un agitador

Para la selección del transistor empleado en circuito de la Figura 3.10 es necesario conocer la potencia que el agitador consumirá en condiciones de switcheo. Las variables necesarias para hacer el cálculo de potencia consumida por la carga es el voltaje colecto-emisor del transistor y la corriente mínima a la que opera la carga.

$$P_{switching} = V_{CE(Sat)} I_{MinCarga} \tag{3.3}$$

En la Tabla 3.1 se realiza la comparación de potencias consumida por la carga para cada transistor. En el caso del 2n222 la potencia consumida es de 0.64W que sobrepasa la potencia máxima de 0.5W que provoca que el transistor se recaliente y dañe, por lo tanto, no es útil para la aplicación que se necesita darle.

	$\mathbf{V_{CE(Sat)}}$ [V]	$\mathbf{I_{MinCarga}}$ [mA]	$\mathbf{P_{Max}}$[W]	$\mathbf{P_{Consumida}}$ [W]
2N2222	1.6	400	0.5	0.64
TIP31	1.2	400	40	0.48

Tabla 3.1 Comparación de transistores 2n222 y TIP31

El consumo de potencia de la carga con el TIP31 es de 0.48W que va de sobra con respecto a su potencia máxima de 40W y se adapta a los requerimientos del circuito de la Figura 3.8.

Otro elemento del sistema de medición que se debe controlar es la electroválvula, y, ya que esta también necesita únicamente de activación y apagado se usa el mismo esquema, con la diferencia de que la electroválvula necesita un voltaje de 12V para funcionar. Por esa razón el esquema electrónico es el siguiente.

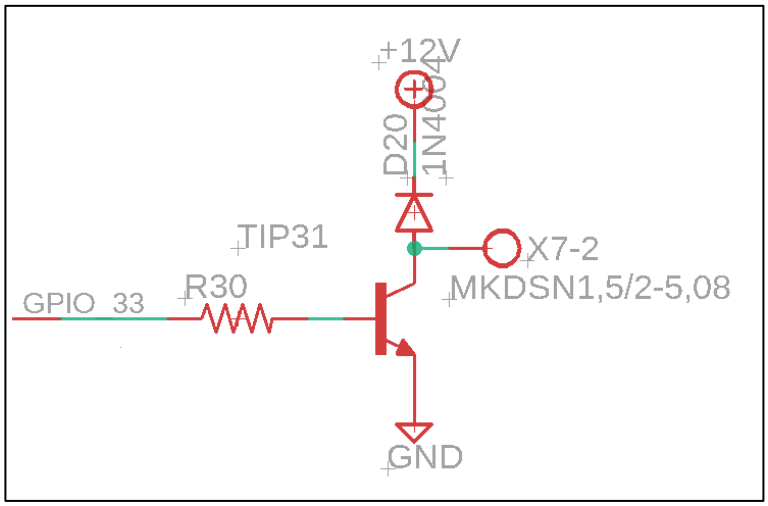

Figura 3.11 Esquema electrónico de conexión de la Electroválvula

Diseño se interfaz electrónica para bomba peristáltica

El siguiente desafío es el control de las bombas peristálticas que serán usadas para encaminar los líquidos, en su dosificación de volumen correcta, al recipiente que contendrá la solución homogénea, es decir contendrá la muestra mezclada con los solutos. La bomba peristáltica opera a 12V y 100mA, por lo que será necesario hacer el diseño de un circuito que permita elevar el nivel de corriente y a su vez opere en frecuencias con valores alrededor de 2kHz y 4kHz, ya que se planea realizar un control mediante PWM.

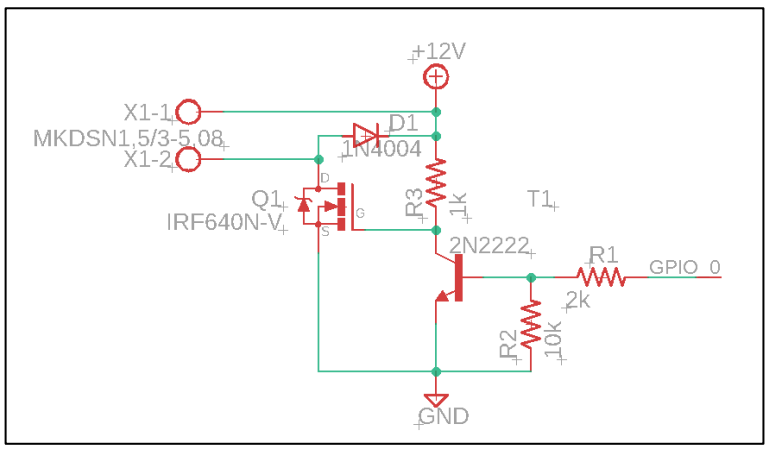

Figura 3.12 Esquema electrónico para el control de una Bomba peristáltica

Una señal PWM es la que permite el accionar de la bomba en un constante switcheo, permitiendo controlar su velocidad y en consecuencia controlar el flujo de líquido que pasa a través de ella. Para el uso de PWM es necesario un

elemento de conmutación rápida electrónico y en el mercado se puede encontrar un sin número de transistores útiles, uno de ellos es el 2n2222.

El uso de un solo transistor no es suficiente para accionar la carga (bomba peristáltica), la bomba opera con 12VDC y corriente de 100mA demandando una potencia de 1.2W, por ello es necesario de un transistor de potencia que permita elevar al nivel de voltaje necesario para el correcto funcionamiento de la bomba. El transistor elegido es el mosfet IRF640, el cual es un transistor de potencia capaz de manejar corrientes de 16 A y tiene una potencia de disipación de hasta 150W. Al ponerlo en cascada conectando la compuerta al colector se cumple con la etapa de potencia necesaria para el accionar de la bomba. La configuración es correcta ya que el voltaje típico de compuerta (V_{GS}) es 3V, que es el nivel de tensión entregado por la FPGA.

Diseño completo de interfaces electrónicas

El esquema con todos los circuitos que controlan los actuadores que manejara el dispositivo experimental se ve reflejado en la Figura 3.13 que fue realizado en la plataforma Eagle que permite realizar el diseño de circuitos electrónicos y PCB.

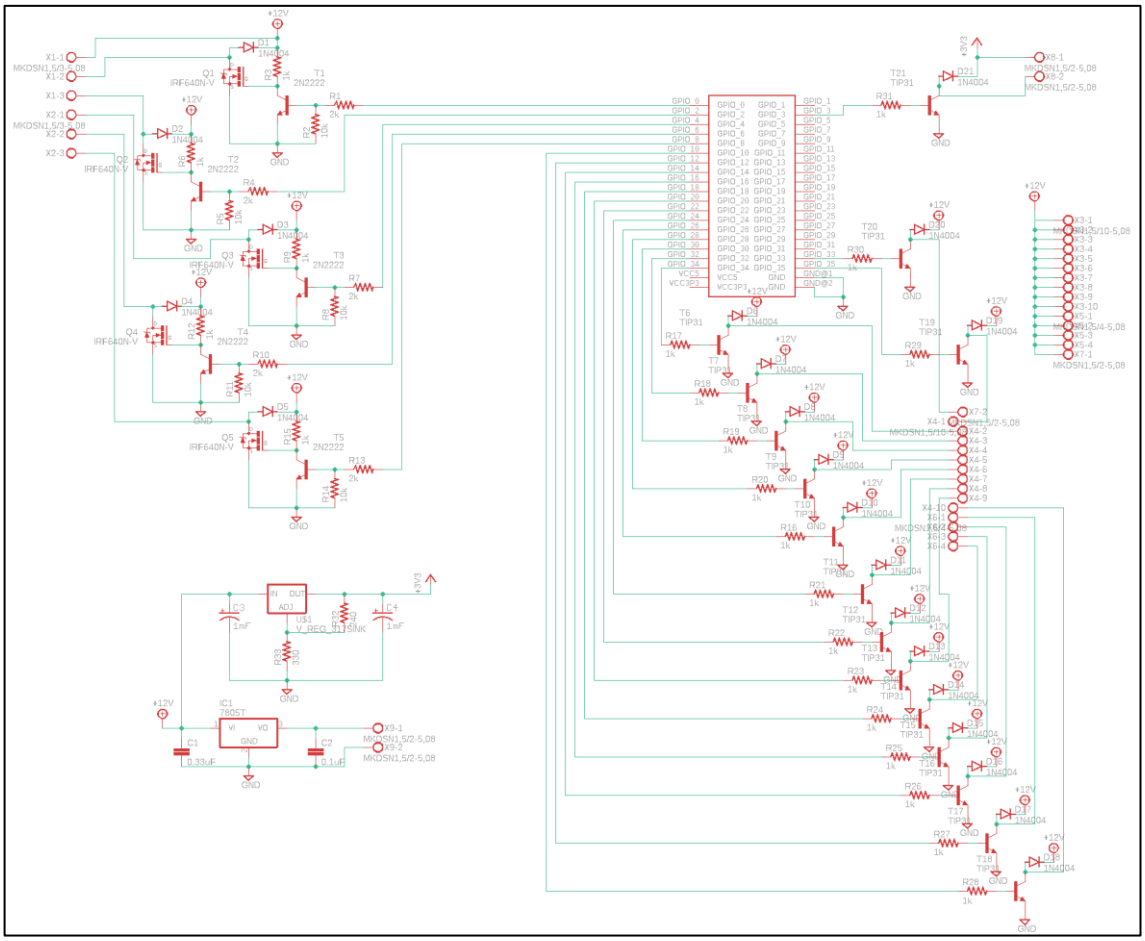

Figura 3.13 Esquema completo para el manejo de periféricos en EAGLE

3.4. Diseño del sistema digital programado en Quartus para la interconexión entre el FPGA y el sistema operativo del mismo.

Para controlar los dispositivos a través de las interfaces electrónicas previamente diseñadas, se necesita de un sistema o circuito digital programado dentro del FPGA, que sirva como interfaz de conexión entre lo electrónico y el procesador del sistema operativo, para que de esta forma el Sistema Operativo pueda procesar los datos.

Las tarjetas de desarrollo FPGA de interés para este proyecto tienen su arquitectura dividida en dos partes esenciales. La primera dedicada al desarrollo de sistemas digitales programables llamada FPGA y la segunda dedicada a procesamiento fuerte de datos llamada HPS. En esta última es donde se encuentra el Sistema Operativo. Estas dos partes no comparten una interconexión lógica directa, así que debe ser programada dentro del FPGA e

informarle al Sistema Operativo de dichas conexiones, las cuales son llamadas Puentes/Bridges.

Las entradas y salidas Físicas de la FPGA no se encuentran directamente conectadas al HPS, esto se debe a que los datos se deben primero procesar en un sistema digital y solo el resultado final llegar al Sistema Operativo por esas razones, el diagrama de bloques de conexión total esperado para el controlador del dispositivo de medición de nitritos se muestra a continuación:

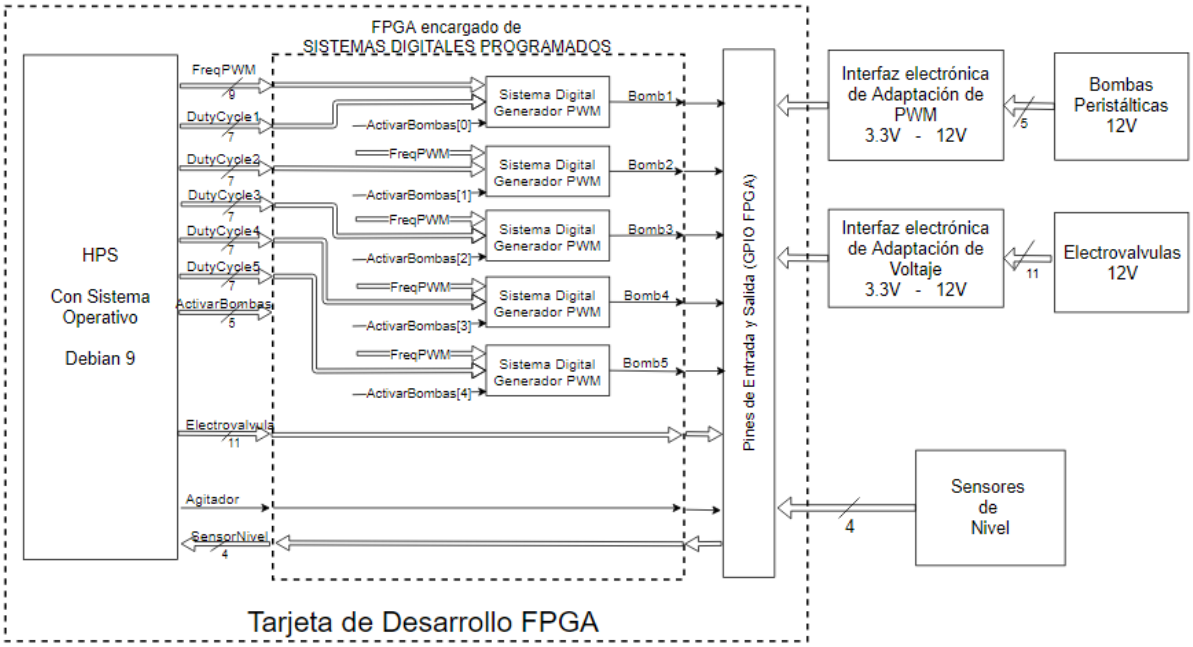

Figura 3.14 Diagrama de Bloques del sistema controlador.

Usando el software de Intel llamado QUARTUS PRIME, en conjunto con lenguajes de programación usados para descripción de hardware, como VHDL y verilog, se describen las conexiones y bloques contenidos en el bloque **Tarjeta de Desarrollo FPGA,** mientras que lo contenido en el bloque HPS y sus conexiones se realizan en la herramienta **Platform Designer** de QUARTUS usada para inicializar el HPS y sus conexiones externas, llamadas puentes (bridges).

3.4.1. Bloque Generador de PWM

Este bloque se encarga de generar la señal PWM para el control de las bombas peristálticas, y, es un sistema digital sincrónico, es decir, depende de un clock. Ya que, gracias a este puede realizar la cuenta de la cantidad de pulsos de reloj

necesarios para obtener la frecuencia y el Duty Cycle deseado en la salida PWM. Dentro de este bloque se realiza una división entre el clock de entrada y la frecuencia de clock deseada para el PWM para obtener la cantidad de pulsos del clock de entrada que se deben contar para cambiar de estado el clock del PWM. Las señales que este bloque usa se listan a continuación:

- Resetn: Al activar esta señal todas las salidas del Generador PWM son apagadas.
- Clock: Al ser un circuito secuencial usa un clock como una señal que determina el instante en que se debe realizar un cambio.
- FreqPWM[8..0]: A través de esta señal de 9 bits se indica al sistema la frecuencia deseada para el PWM.
- En: Mediante esta señal, se activa la señal PWM en la salida PWMSignal. Si “en” no está activa entonces la señal PWM es apagada.
- ChargeFreq: Usando esta señal se indica al sistema que debe realizar las operaciones necesarias para cambiar la frecuencia del PWM a la frecuencia presente en la señal “FreqPWM[8..0]”. El fin de esta operación es indicado por la salida “finCharge”.
- DutyCycle[6..0]: En esta señal de 9 bits se indica el Duty Cycle deseado para la señal PWM.
- PWMSignal: En esta salida está presente la señal PWM que controla las bombas.
- FinCharge: Esta señal indica el fin de la operación de carga de frecuencia, como se indicó anteriormente.
- ClockPWM: Esta señal de salida tiene el clock con el que el PWM realiza la cuenta para el Duty Cycle, es decir, 100 pulsos de este reloj es un ciclo del PWM. Entonces, por ejemplo, si se desea obtener un Duty Cycle del 70% se mantendra encendida “PWMSignal” durante 70 pulsos del reloj “ClockPWM” y los restantes 30 pulsos se deberá apagar dicha señal.

El diagrama de entradas y salidas del bloque descrito se muestra a continuación:

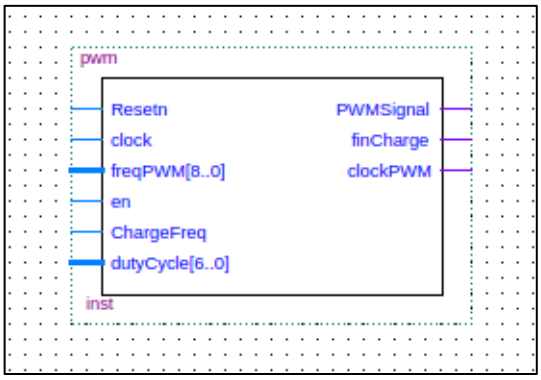

Figura 3.15 Diagrama de entradas y salidas del bloque generador de PWM. Izquierda entradas y Derecha salidas

3.4.2. Instanciación HPS

La instanciación del HPS se realiza en el Platform Designer de Quartus en el cual también se definen las memorias, puertos y bridges que el sistema tendrá; todo esto queda almacenado en un archivo “Qsys”. En la imagen a continuación se muestra un resumen del sistema diseñado para el presente proyecto.

Use	Co...	Name	Description	Export	Clock	Base	End	IRQ
☑		hps_0	Arria V/Cyclone V Hard Processor System		*multiple*	0x0000_0000	0xffff_ffff	
☑		hps_only_ma...	JTAG to Avalon Master Bridge		clk_0			
☑		sysid_qsys	System ID Peripheral Intel FPGA IP		clk_0	0x0001_0000	0x0001_0007	
☑		onchip_memory...	On-Chip Memory (RAM or ROM) Intel FPGA IP		clk_0	0x0000_0000	0x0000_0fff	
☑		jtag_uart	JTAG UART Intel FPGA IP		clk_0	0x0002_0000	0x0002_0007	0
☑		fpga_only_ma...	JTAG to Avalon Master Bridge		clk_0			
☑		intr_capturer_0	Interrupt Capture Module		clk_0	0x0003_0000	0x0003_0007	1
☑		clk_0	Clock Source		*exported*			
☑		PWMFreq	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_00d0	0x0000_00df	
☑		DutyCycle1	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_00c0	0x0000_00cf	
☑		DutyCycle2	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_00a0	0x0000_00af	
☑		DutyCycle3	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_0090	0x0000_009f	
☑		DutyCycle4	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_0080	0x0000_008f	
☑		DutyCycle5	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_0070	0x0000_007f	
☑		ControlPWM	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_00b0	0x0000_00bf	
☑		controlBomb	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_0020	0x0000_003f	
☑		Agitador	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_0060	0x0000_006f	
☑		ElectroValvula	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_0000	0x0000_001f	
☑		SensorNivel	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_0050	0x0000_005f	2
☑		Absorbancia	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_0040	0x0000_004f	

Figura 3.16 Resumen del Sistema Qsys del controlador

En el diagrama presentado en la figura anterior se muestran los elementos del sistema y su rango de direcciones ubicado entre Base y End, las cuales son direcciones de memoria usadas para que el sistema operativo del HPS pueda acceder y controlar los bridges necesarios. Los elementos usados en este sistema se listan a continuación:

- Hps_0: este es el procesador o el núcleo del sistema, que, como se ve en descripción es un procesador Cyclone V. Se decidió elegir entre este tipo de procesadores, puesto que, las tarjetas que incluyen procesadores más potentes como el Arria 10 y el Stratix 10 tienen un precio elevado y sobre

exceden lo necesario para este proyecto. Las diferencias entre estos procesadores se listan en la tabla a continuación.

HPS Module	Cyclone V SoC	Arria V SoC	Arria 10 SoC	Stratix 10 SoC
Microprocessor Unit Subsystem (MPU)	Single / Dual Cortex®-A9	Dual Cortex-A9	Dual Cortex-A9	Quad Cortex-A53
Cache Coherency Controller	Accelerator coherency port (ACP)	ACP	ACP	Cache Coherency Unit (CCU)
System Memory Management Unit	No	No	No	Yes
On-Chip RAM	64 KB	64 KB	256 KB	256 KB
Error Correction Code (ECC) Controller	No	No	Yes	Yes
HPS-FPGA Bridges	Yes	Yes	Yes	Yes
General Purpose I/O (GPIO)	Yes	Yes	Yes	Yes
Available dedicated I/Os	10	10	17	48
Available shared I/Os	Up to 67	94	48	0
SDRAM Controller	Inside HPS	Inside HPS	Outside of HPS	Outside of HPS
NAND Flash Controller	Yes	Yes	Yes	Yes
Secure digital/multimedia card (SD/MMC) Controller	Yes	Yes	Yes	Yes

MPU Subsystem Feature	Cyclone V SoC	Arria V SoC	Arria 10 SoC	Stratix 10 SoC
CPU	Single/Dual Cortex-A9	Dual Cortex-A9	Dual Cortex-A9	Quad Cortex-A53
Maximum frequency (MHz)	925	1050	1500	1500
Core revision	r3p0	r3p0	r4p1-00rel0	r0p4-51rel0
L1 instruction cache	32 KB	32 KB	32 KB	32 KB
L1 data cache	32 KB	32 KB	32 KB	32 KB
L2 cache	512 KB	512 KB	512 KB	1 MB
ACP enabled	Yes	Yes	Yes	No (2)
L1 data cache error checking	Parity (3)	Parity(3)	Parity(3)	ECC (4)
L1 instruction cache error checking	Parity(3)	Parity(3)	Parity(3)	Parity(3) on data and tag bits
L2 cache error checking	ECC(5) ; ECC interrupts; optional parity(3) for tag bits	ECC(5); ECC interrupts; optional parity (3) for tag bits	ECC(5); ECC interrupts; optional parity (3) for tag bits	ECC(5) on data and tag bits; ECC interrupts
Translation lookaside buffer (TLB) error checking	Parity(3)	Parity(3)	Parity(3)	Parity(3)

Figura 3.17 Diferencias entre Hard Processors usados en SOC FPGA's *[15]*

- Hps_only_master: este es un puente entre el puerto JTAG, usada para la carga del programa FPGA, y el sistema operativo.
- SySid_qsys: Bloque que asigna un id de identificación al sistema, y, siempre es requerido.
- Onchip_memory: Bloque de memoria usado para asignar RAM al sistema operativo.

- Jtag_uart: Activa el puerto UART para la comunicación del sistema operativo con un Host PC, esto permite realizar la configuración inicial del OS.
- Fpga_only_master: este es un puente entre el puerto JTAG de la FPGA y el sistema de memoria del HPS.
- Intr_capturer: Encargado de capturar las interrupciones que los demás bloques realicen.
- Clk_0: Fuente de reloj de 5MHz, basándose en el clock del HPS, el cual es de 50 MHz.
- PIO (Parallel Input/Output): Bridges creados para comunicar el sistema operativo con el sistema digital.
    - PWMFreq[8..0]: Puente/Bridge de 9 bits mediante el cual el HPS indica al sistema digital la frecuencia deseada para el PWM que controla las bombas.
    - DutyCycle1, DutyCycle2, DutyCycle3, DutyCycle4, DutyCycle5: Puentes de 7 bits mediante los cuales el HPS indica al sistema digital que Duty Cycle necesita para cada bomba.
    - ControlPWM: Mediante esta señal de 1 bit se le indica al bloque PWM que realize la carga de frecuencia deseada para el PWM.
    - ControlBomb[4..0]: Puente de 5 bits mediante el cual el HPS indica que salida de PWM activar o desactivar. Cada bit representa una bomba, es decir, si el bit correspondiente a una bomba está en ‘1’, entonces, dicha bomba se encenderá, caso contrario, permanecerá apagada.
    - Agitador: Puente de 1 bit usado para indicar el estado deseado para el agitador, ya sea, encendido o apagado.
    - Electrovalvula[14..0]: Puente de 15 bits mediante el cual se activan las electroválvulas. Cada bit representa una válvula, es decir, si el bit correspondiente a una válvula está en '1', entonces dicha válvula se encenderá, caso contrario, permanecerá apagada.
    - Sensor Nivel [3..0]: Puente de 4 bits mediante el cual se lee el estado de los sensores de Nivel, este sensor mide si existe liquido presente en un contenedor. Cada bit representa un sensor de nivel.

- Absorbancia [7..0]: Puente de 8 bits a través del cual el colorímetro comunicara a la FPGA la medida de absorbancia.

3.4.3. Sistema Digital Completo

El sistema digital completo se encarga de describir todas las conexiones presentes en el bloque llamado “Tarjeta de Desarrollo FPGA” en la Figura 3.13. Estas conexiones son escritas en el lenguaje de descripción de hardware Verilog, por su versatilidad en el diseño de sistemas digitales. El código se encuentra adjunto en Anexos.

La compilación de este archivo es el programa que se carga en la FPGA, esta compilación nos da información sobre los recursos del FPGA que consume el sistema diseñado. Este informe se muestra en la Figura 3.17 a continuación.

Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Standard Edition
Revision Name	soc_system
Top-level Entity Name	ghrd
Family	Cyclone V
Logic utilization (in ALMs)	N/A
Total registers	5207
Total pins	230
Total virtual pins	0
Total block memory bits	34,816
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	1

Figura 3.18 Recursos utilizados por el sistema digital diseñado

Entre los FPGA’s Cyclone V de menor coste que ofrece Intel y que cumplen estos requerimientos se encuentran los siguientes:

FPGA Device	De1-SOC [16]	De0-Nano-SOC [17]	DE10-Nano [18]	DE10-Standar [19]
Precio	$ 175	$ 90	$ 110	$ 259
Tamaño	166*130 mm	68.6 * 96 mm	68.6 * 107 mm	166 * 130 mm
Procesador	Cyclone V SoC 5CSEMA5F31C6	Cyclone® V SE 5CSEMA4U23C6N	Cyclone® V SE 5CSEBA6U23I7NDK	Cyclone V SoC— 5CSXFC6D6F31C6N
Logic Elements (LE's)	85 K	49 K	110 K	110 K
ALM's	32070	15880	41910	41910
Memory Bits	4,450 Kbits	3080 Kbits	5570 Kbits	5662 kbits
Memoria RAM	GB (2x256Mx16) DDR3 SDRAM	1GB DDR3 SDRAM (32-bit data bus)	1GB DDR3 SDRAM (32-bit data bus)	1GB (2x256Mx16) DDR3 SDRAM on HPS
Total Pins	457	314	326	499

Tabla 3.2 Comparación entre FPGAs de bajo precio

Como se puede observar en la tabla el dispositivo de menor coste y que cumple los requerimientos deseados es la tarjeta de desarrollo DE0-nano SOC y es en esta en la cual se trabajara el presente proyecto. El resultado de la compilación para la tarjeta DE0-nano SOC se muestra a continuación, en donde, también se puede observar el porcentaje de recursos utilizados.

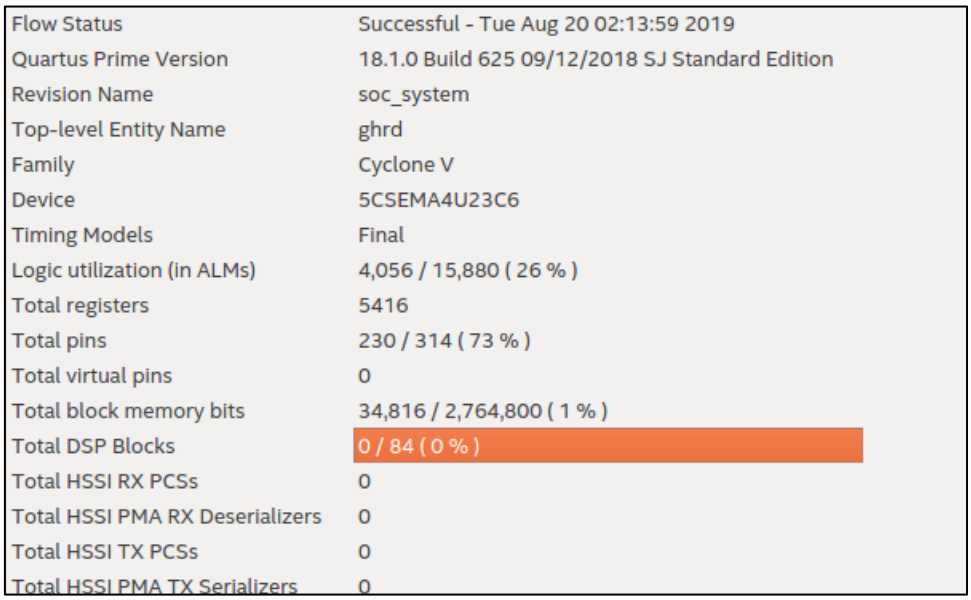

Figura 3.19 Compilación del Sistema Digital para la tarjeta de desarrollo DE0-nano SOC

3.5. Diseño del software controlador

El HPS corre un sistema operativo instalado en una memoria microSD. Para este proyecto el OS a usar es Debian 9, ya que, este cuenta con la facilidad de tener disponibles las librerías y dependencias necesarias para programar en lenguaje C.

El manejo de los bridges/puentes entre el HPS y el sistema digital en el Sistema Operativo, se basa en lectura y escritura de direcciones de memoria. Estas direcciones de memoria se definen en el archivo Qsys creado en Plattform Designer explicado en la sección anterior.

Con la ayuda de Quartus se puede crear una librería para lenguaje C, llamada hps_0.h, con un resumen de todas las direcciones de memorias presentes en el archivo Qsys. Dentro de hps_0.h las direcciones de memoria en donde inician los puentes se nombran de la siguiente manera:

- PWMFreq_Base
- DutyCycle_Base
- ControlPWM_Base
- ControlBomb_Base
- Agitador_Base
- ElectroValvula_Base
- SensorNivel_Base
- Absorbancia_Base

La dirección de memoria importante es el inicio o base dado que cada bloque o puente, bloque PIO (Parallel Input/Output), tiene un mapa de registros los cuales controlan las operaciones que realiza. El mapa de registros de un PIO se muestra a continuación en la Figura 3.20.

Offset	Register Name		R/W	(n-1)	...	2	1	0
0	data	read access	R	Data value currently on PIO inputs				
		write access	W	New value to drive on PIO outputs				
1	direction *(1)*		R/W	Individual direction control for each I/O port. A value of 0 sets the direction to input; 1 sets the direction to output.				
2	interruptmask *(1)*		R/W	IRQ enable/disable for each input port. Setting a bit to 1 enables interrupts for the corresponding port.				
3	edgecapture *(1), (2)*		R/W	Edge detection for each input port.				
4	outset		W	Specifies which bit of the output port to set.				
5	outclear		W	Specifies which output bit to clear.				

Figura 3.20 Mapa de registros de un PIO (Parallel Input/Output)

Por esta razón cada variable que represente un bridge en el código debe ser declarada como un puntero a la dirección de memoria especificada por la base, y, dependiendo de la operación a realizar se le puede añadir un offset a dicho puntero.

Todo esto se debe considerar al realizar el proyecto en C, para lo cual se necesita acceder al sistema operativo, esto se logra a través del puerto UART de la FPGA el cual lleva la siguiente configuración de puerta serial:

Baud Rate	115200
Bits de Datos	8
Paridad	None
Bits de Parada	1

Tabla 3.3 Configuración de puerta serial de la FPGA

Una vez dentro del sistema operativo se crea el proyecto en lenguaje C, y se ejecuta la rutina especificada en la

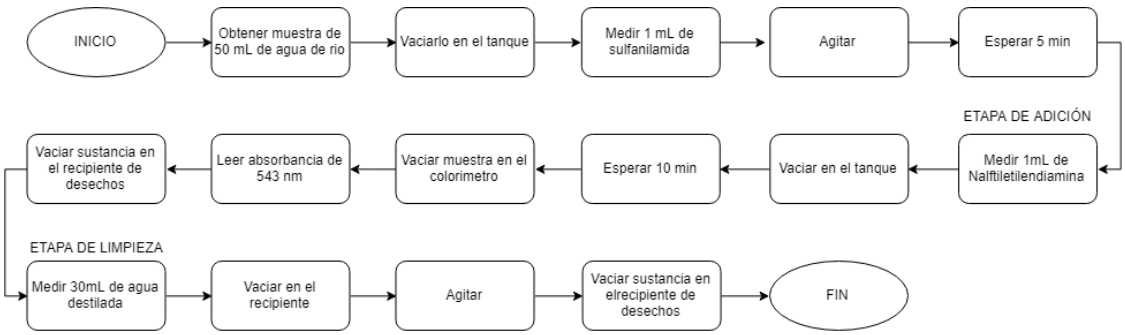

Figura ***3.4***, en donde se específica el diagrama de flujo del proceso que debe realizar el dispositivo de medición de NO2. Una parte del proceso es tomar la lectura de absorbancia proveniente del colorímetro a una longitud de onda de 543nm, la cual, posteriormente debe convertirse a un valor de concentración. Para lograr esta conversión se sabe que la Ley de Beer dice que la absorbancia

y la concentración están linealmente relacionadas [20]. Por lo que, se toman varias muestras a concentraciones conocidas y se determina su absorbancia, para luego linealizar estos datos y determinar la ecuación que los represente.

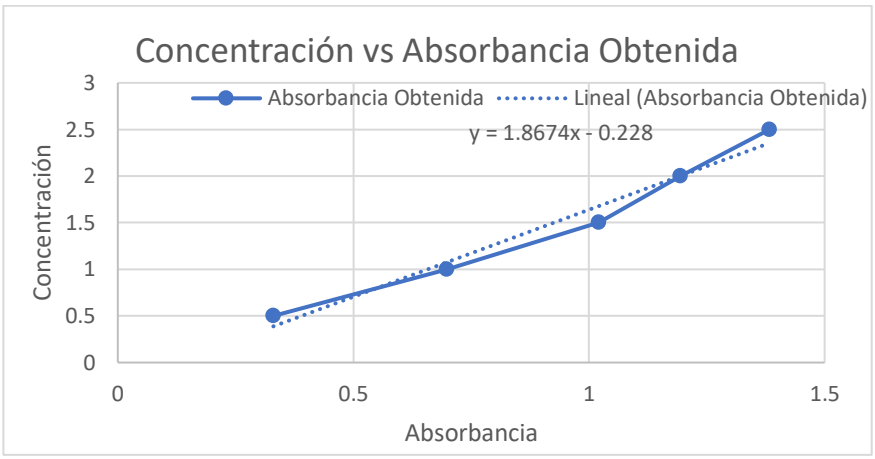

Figura 3.21 Linealización de los datos para obtener una ecuación que los represente

En la Figura 3.21 se muestra la linealización de los datos obtenidos y la ecuación que relaciona la absorbancia con la concentración es la siguiente:

$$C = 1.8674A - 0.228$$

Donde C es la concentración de nitritos y A es la absorbancia medida. El código en lenguaje C con toda la rutina completa y sus dependencias se encuentra en Anexos.

El algoritmo de control es diseñado para controlar los bridges usando las direcciones de memoria descritas en el archivo hps_0.h como se indicó anteriormente. Todo esto es realizado como un proyecto escrito en lenguaje C el cual usa los siguientes archivos:

- fpga.h: Este archivo es usado como librería, es decir, contiene todos los nombres de las funciones que usara el main.c.
- fpga.c: Este archivo es usado para desarrollar el código de cada función presente en fpga.h.
- main.c: Este es el programa principal, en donde se encuentra descrito el algoritmo de control.
- MakeFile: Aquí se indica las propiedades de compilación del proyecto.

3.6. Análisis de Datos

Para tener una idea de si el procedimiento se está realizando correctamente y validar que el valor arrojado por el dispositivo sea preciso; el proceso de lectura de concentración de NO2 se realiza 3 veces sobre la misma muestra, y, con los 3 datos obtenidos se realiza un análisis de desviación estándar, en donde, si el valor obtenido de desviación es muy grande significa que los datos fueron erróneos y que el dispositivo necesita ser revisado, calibrado o necesita mantenimiento.

Se hace de esta forma, debido a que, si existiera algún dato aberrante, la desviación estándar crecerá, logrando detectar fallos y tener mejores medidas de concentración de nitratos.

CAPÍTULO 4

4. ANÁLISIS DE RESULTADOS

En este capítulo se muestran los resultados de la evaluación del sistema de control basado en FPGA, luego se describen las pruebas realizadas con el prototipo de control y finalmente un análisis de los costos y ventajas del diseño realizado.

4.1. ANÁLISIS DEL FUNCIONAMIENTO DE INTERFACES ELECTRÓNICAS.

Para evaluar el funcionamiento del esquema electrónico se probó el funcionamiento de cada circuito en Protoboard, la prueba consistía en comprobar si el diseño era capaz de controlar los elementos de forma individual usando señales de 3.3V de entrada, ya sea, activando o desactivando elementos, o, controlando flujo de bombas por PWM

Para el circuito encargado de activar y desactivar los agitadores y electroválvulas se armó el circuito en protoboard, y se conectó los elementos a la salida del circuito, luego mediante un switch se pasaba entre tener en la entrada del circuito una fuente de 3.3V o tierra. Con esto se logró comprobar que el circuito era capaz de encender, apagar y manejar la potencia requerida por los elementos, ya que, al mantener encendido los elementos por 10 minutos se comprobó que los transistores usados no calentaban en exceso incluso sin disipadores. Este calentamiento no es problema para el sistema controlador, puesto que las electroválvulas se encienden durante cortos periodos de tiempo y con largos tiempos de reposo.

Para el circuito encargado de manejar las bombas se probó su funcionamiento en protoboard, conectando las bombas a la salida, mientras que, en la entrada del circuito se conectó un generador de PWM de 3.3V. Luego, se realizó variaciones del Duty Cycle y de la frecuencia comprobando que la velocidad de las bombas era controlada y por tanto el flujo de líquido que pasa por ellas puede ser controlado. De estas pruebas se obtuvo que la frecuencia ideal para el control de las bombas es de 2 KHz.

Una vez comprobado el funcionamiento de los circuitos, se realizó el diseño del circuito impreso, obteniendo una PCB de 17.5 cm x 10.9 cm como se puede apreciar en la Figura 4.1.

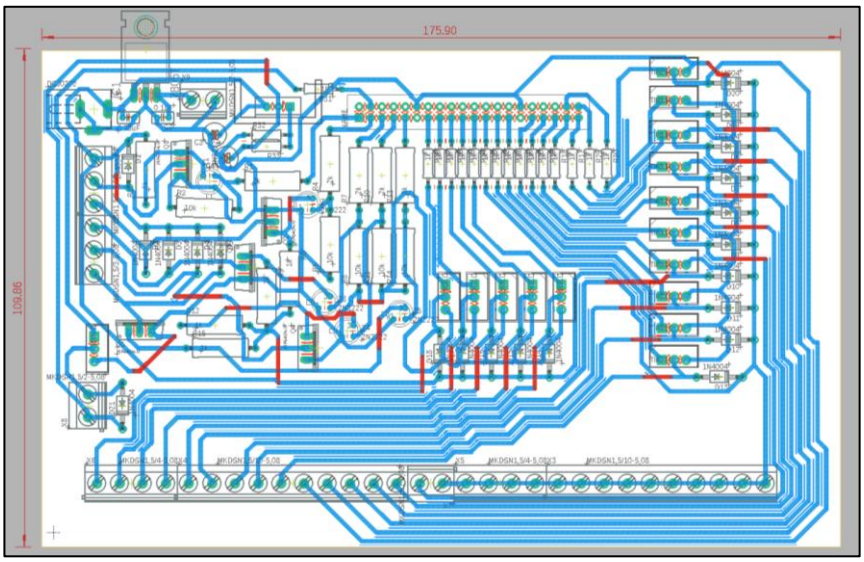

Figura 4.1 Diseño electrónico de placa en PCB utilizando Eagle

Tomando en cuenta el número de actuadores que van a ser conectados a la placa y que actualmente en el país la microelectrónica no se encuentra muy desarrollada, se hizo uso de la electrónica tradicional dando como resultado una placa de tamaño considerable debido a la cantidad y tamaño de sus componentes pero que cumple con su propósito de forma eficaz.

A continuación, se muestra el resultado del diseño electrónico ya montado en la placa de fibra de vidrio en la Figura 4.2.

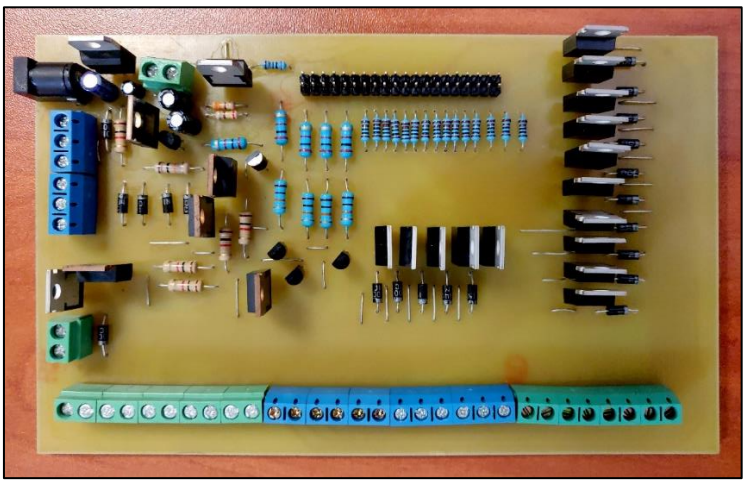

Figura 4.2 Resultado final del montaje del circuito electrónico en PCB

El espacio que ocupara el controlador (FPGA) y la placa de acondicionamiento electrónico es muy importante**.** Por lo que con el fin de usar el menor espacio posible se optó por colocar la FPGA encima de la placa, reduciendo su espacio al máximo. En la Figura 4.3 se puede apreciar cómo se realizó el montaje desde una vista superior.

Con esta colocación se tiene todo integrado en un espacio más reducido y permite que las conexiones con los actuadores se puedan realizar de forma rápida y ordenada.

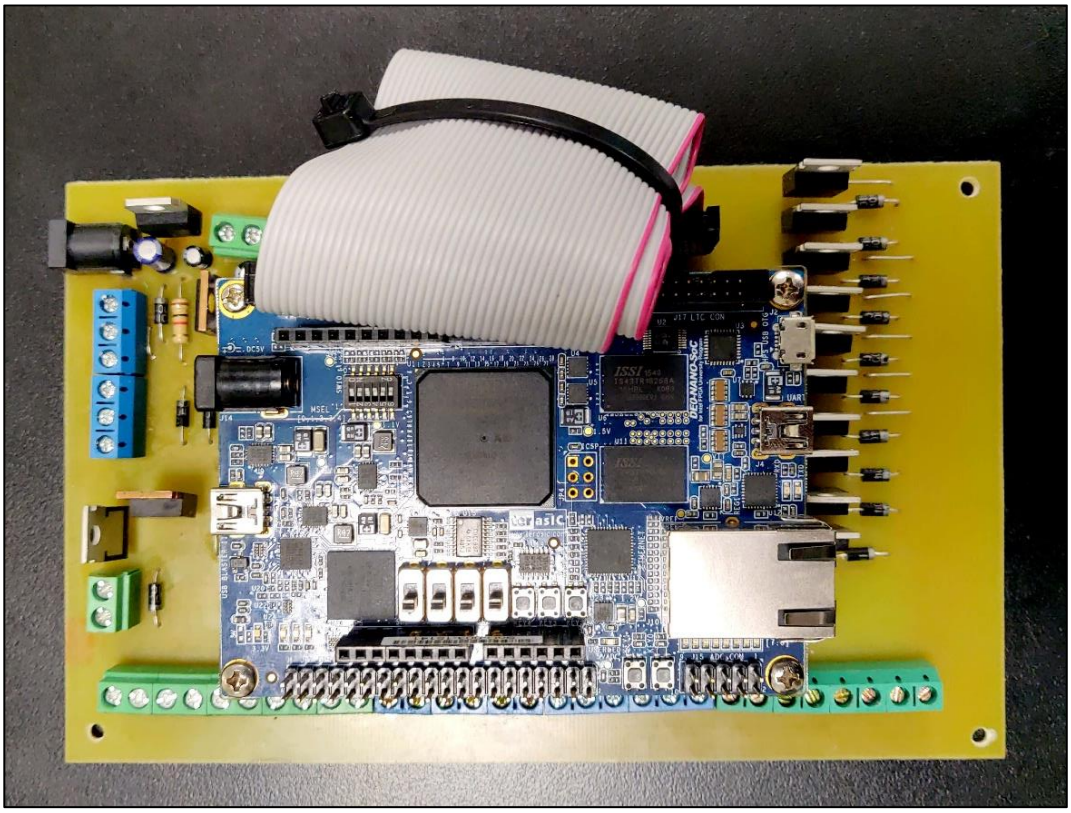

Figura 4.3 Montaje de la placa junto a la FPGA vista superior

El FPGA fue ubicado a una altura considerable de tal forma que el calor generado por la tarjeta y la placa pueda circular. En la Figura 4.4 se puede apreciar que también posee una cobertura de acrílico que sirve como aislante, con el fin de que no se produzcan cortocircuitos por algún tipo de contacto entre la tarjeta y la placa.

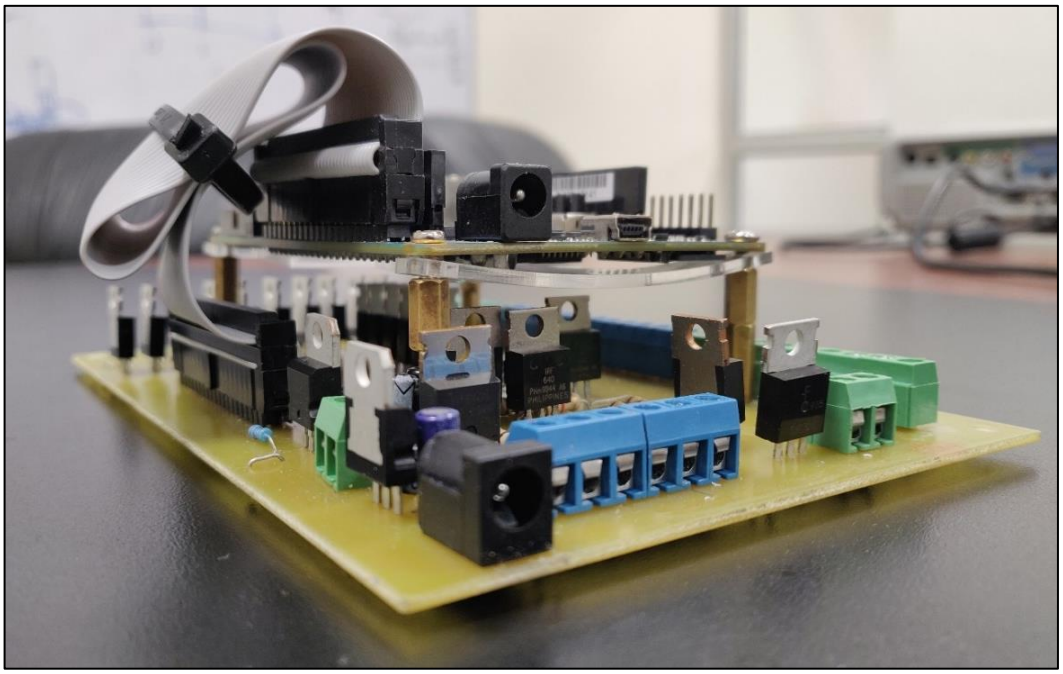

Figura 4.4 Montaje de la placa junto a la FPGA vista lateral

4.2. SALIDA SERIAL PARA EL MONITOREO DEL PROCESO.

El algoritmo de control basa su funcionamiento en el diagrama de flujo presente en la Figura 3.4. Este describe paso a paso el cómo se realizan las mezclas para obtener la solución homogénea coloreada, y además implementa la rutina para convertir la absorbancia en concentración. Además, dado que el sistema basa sus resultados en un proceso de mezclas químicas y estas pueden llegar a contaminarse se implementa un análisis de datos basado en desviación estándar, que consiste en la toma de 3 datos consecutivos a la misma muestra, e identificar si la desviación entre los 3 datos es normal o demasiado grande.
Este sistema cuenta con una salida serial para el monitoreo del proceso, la cual se puede notar en la Figura 4.5

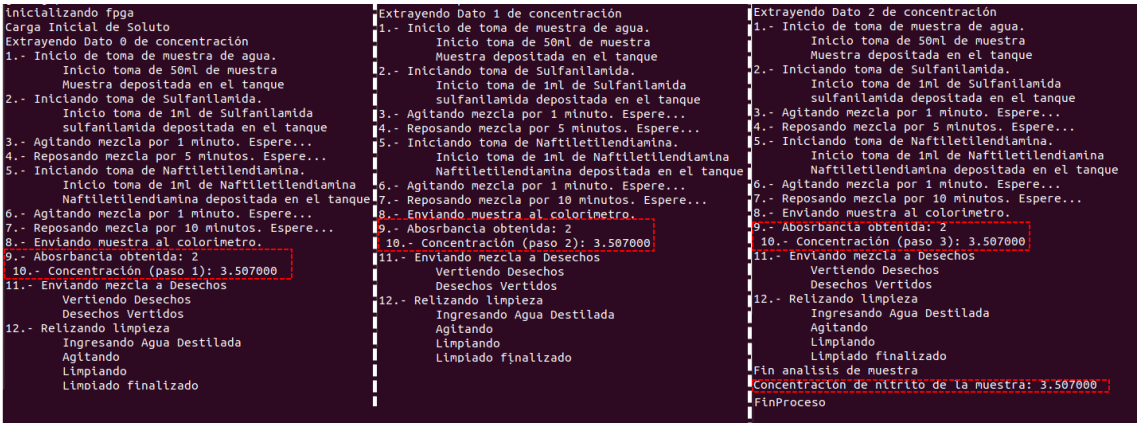

Figura 4.5 Salida de monitoreo del programa para el control del dispositivo de medición de nitritos

Usando este algoritmo se realizaron varios experimentos para determinar los tiempos de activación y desactivación de las bombas y electroválvulas para lograr una mejor dosificación volumétrica. Además, se estudió el cómo influenciaba el control PWM en la obtención de los líquidos. El resultado obtenido de estos experimentos se muestra en la sección 4.4.

4.3. ANÁLISIS DE COSTOS

Una vez implementado por completo el controlador del dispositivo de medición de nitrito se obtiene el valor total de la propuesta de reemplazar un PLC por una FPGA.

Tal como se lo esperaba, el precio estimado de la FPGA y la placa de acondicionamiento electrónico suma un valor de $158.30.

A continuación, en la Tabla 4.1 se detalla cada uno de los materiales y componentes usados para la implementación del nuevo controlador.

CANTIDAD	VALOR	DISPOSITIVO	PRECIO	IMPORTE
1	0.1uf	Capacitor	0.10	0.10
1	0.33uf	Capacitor	0.35	0.35
2	1uF	Capacitor	0.40	0.80
5	10k	Resistencia 1/2W	0.10	0.50
5	2k	Resistencia 1/4W	0.05	0.25
16	1k	Resistencia 1/4W	0.05	0.80
1	240	Resistencia 1/4W	0.05	0.05
1	330	Resistencia 1/4W	0.05	0.05
21	1N4004	Diodo	0.15	3.15
5	2N2222	Transistor	0.20	1.00
1	7805T	Regulador de voltaje 5V	0.75	0.75

1	LM317	Regulador de voltaje 3V	0.65	0.65
40		Espadines macho	0.50	20.00
5	IRF640	Mosfet	0.4	2.00
16	TIP31	Transistor	0.50	8.00
3	T-Block 3	Borneras	0.25	0.75
7	T-Block 2	Borneras	0.25	1.75
1		Adaptador de 12V a 5ª	6.00	6.00
1		Bus de datos	1.00	1.00
1		Conector Arduino	0.25	0.25
1	2.1MM	Adaptador Jack DC	0.60	0.60
1	17.5 x 10.9 CM	Placa en fibra	30.50	30.50
1	DC0 NANO	FPGA	79.00	79.00
TOTAL				158.30

Tabla 4.1 Gasto total de controlador para dispositivo de medición con FPGA

El dispositivo contaba con un controlador PLC MicroLogix 1100 y con un módulo de expansión cuyo valor en conjunto asciende a $584.89. Este valor es aproximadamente el triple del valor considerado por la propuesta usando FPGA. A continuación, en la Tabla 4.2 se detalla los precios y modelos usados.

CANTIDAD	DISPOSITIVO	PRECIO	IMPORTE
1	MicroLogix 1100	398.40	398.4
1	Módulo de expansión MicroLogix 1762	186.49	186.49
TOTAL			584.89

Tabla 4.2 Gasto total de controlador para dispositivo de medición con PLC

4.4. Prueba de precisión

El volumen de líquido que entrega el dispositivo es de suma importancia ya que se están tomando en consideración reacciones químicas de cada uno de los solutos en la muestra de agua. El volumen ideal que debe entregar el dispositivo es 1 mL, pero en la práctica pueden existir perdidas que afecten el proceso químico que realizan los solutos. Este parámetro determina cual será la precisión para la obtención de la concentración de nitritos del sistema de medición, por ello su análisis. A continuación, se muestra en la **Tabla 4.3 Medias obtenidas de**

pruebas con FPGA y PLC con media de referencia de 1mL Tabla 4.3 los resultados obtenidos después de realizar las pruebas tanto en FPGA como en PLC.

	FPGA		PLC	
Nro	SOLUTO 1	SOLUTO 2	SOLUTO 1	SOLUTO 2
1	1.05	0.98	0.95	0.85
2	0.98	1.00	0.85	0.90
3	0.99	1.00	0.90	0.95
4	1.05	1.10	1.05	0.90
5	1.05	1.05	0.90	0.95
6	1.05	1.02	0.90	0.90

Tabla 4.3 Medias obtenidas de pruebas con FPGA y PLC con media de referencia de 1mL

La Figura 4.5 muestra los valores obtenidos en las pruebas con la FPGA de forma gráfica, en donde se puede apreciar que 2 de 6 veces el volumen obtenido para la solución 2 fue exacto, y las 4 pruebas restantes resultaron en valores dispersos cercanos a la medición. Para el soluto 1 el valor más cercano al deseado fue de 0.99mL.

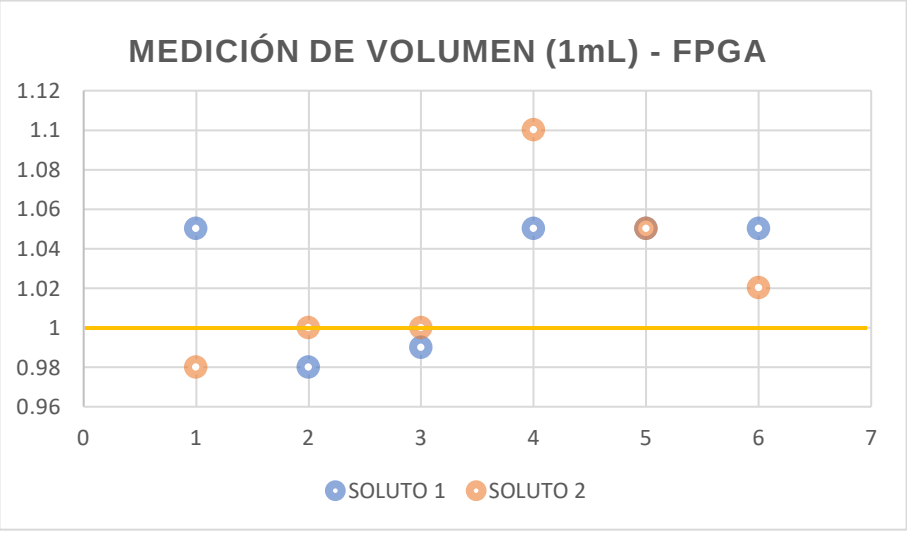

Figura 4.6 Mediciones de soluto 1 y 2 tomando como referencia 1mL con FPGA

La Figura 4.7 presenta los resultados obtenidos usando el PLC y ninguno de los resultados entrego el volumen de 1mL que se requiere. El valor más cercano a 1 mL es de 0.95 mL. Los demás valores oscilan entre 1.05 y 0.85 mL.

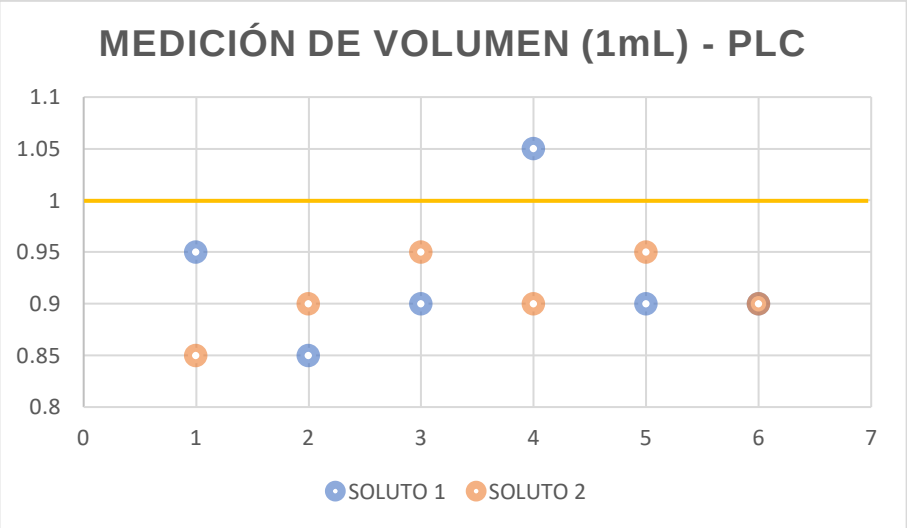

Figura 4.7 Mediciones de soluto 1 y 2 tomando como referencia 1mL con PLC

Comparando los datos obtenidos de ambos controladores se obtiene que la desviación con respecto al valor esperado de 1ml usando PLC es $\sigma_{sol1} = 0.097894$ y $\sigma_{sol2} = 0.097894$; mientras que la desviación respecto al valor de 1mL usando FPGA es $\sigma_{sol1} = 0.041833$ y $\sigma_{sol2} = 0.047081$. Como se puede observar los datos con el controlador basado en FPGA se desvían menos del valor de volumen esperado, lo que indica una buena precisión y además exactitud en la dosificación volumétrica.

CONCLUSIONES

Se reconoció la arquitectura, funcionamiento y parámetros necesarios para que el sistema de medición establezca comunicación y adquisición de datos con el el prototipo de control.

Se analizó el proceso que controla el PLC en el sistema de medición experimental y se crea un algoritmo en el lenguaje de programación C capaz de realizar el mismo proceso con mejoras en la dosificación del volumen.

Se diseñó una interfaz electrónica para interconectar el FPGA y el sistema de medición, permitiendo el manejo de las señales que activan los actuadores, considerando que el sistema de medición actúa bajo niveles tensión y corriente relativamente altas; así se obtuvo un prototipo de controlador que en conjunto con el algoritmo diseñado presentan una mejora en la dosificación volumétrica de los solutos y además permite un análisis basado en la desviación estándar para determinar si la medición es correcta o si el equipo debe ser revisado.

Se comprobó que el uso de la electrónica cotidiana es suficiente para llevar a cabo un proyecto en el que se necesite la implementación de un circuito impreso para reducir costos, en este caso el valor por la implementación del controlador con FPGA es el 27% del valor de controlador con PLC, reduciendo los costos significativamente.

Se diseñó componentes de Software creando drivers que interconectan los elementos eléctricos y electrónicos del sistema de medición con el sistema operativo que corre el FPGA. Estos drivers permiten controlar todo el sistema de medición desde un proyecto en lenguaje C, logrando tener un control sobre todas las variables inmersas en el proceso de medición.

Se evaluó el prototipo de control dando resultados exitosos respecto a la precisión y exactitud de la dosificación volumétrica, tanto de los solutos como de la muestra, comprobando así que el uso del FPGA mejora no solo el proceso de medición, sino que también significa un ahorro de dinero. Además, dada la gran cantidad de entradas y salidas que puede manejar la FPGA brinda escalabilidad al sistema, haciendo posible en un futuro añadir sensores o actuadores, pudiendo llegar a tener un dispositivo completo para el control de la calidad de agua In-Situ.

RECOMENDACIONES

Antes de realizar el diseño es de suma importancia que se estudie a fondo las características eléctricas que tiene cada componente del cual se hará el acondicionamiento, esto le permitirá escoger los elementos correctos para el diseño, evitando que a futuro tenga problemas de recalentamiento por no haber dimensionado el circuito de acuerdo con las exigencias eléctricas del componente.

Al momento de diseñar un circuito tomar en consideración la potencia que consumirá la carga, esto le permite sobredimensionar el circuito de tal manera que el circuito opere con normalidad sin sufrir recalentamientos.

Actualmente el dispositivo toma datos de absorbancia de un microcomputador el cual lee los datos usando un lenguaje de alto nivel y algoritmos complejos que el PLC por sí solo no puede ejecutar. Dado que la FPGA tiene la capacidad de procesamiento suficiente, el algoritmo de lectura de absorbancia puede ser implementado directamente en la FPGA ahorrando el costo del microcomputador.

El uso del FPGA es recomendado para empresas que tienen maquinaria antigua y aún tienen vida útil, pero tienen problemas de compatibilidad para realizar actualizaciones, ya que, el FPGA permite que esa maquinaria adquiera funciones que antes no poseía como comunicación inalámbrica, automatización de proceso, adquisición de datos, conectividad, entre otras que en la industria son de suma importancia. Este es el caso de ECRobotics que busca mejorar sus dispositivos con esta alternativa.

BIBLIOGRAFÍA

[1] J. G. Castro Lugo, J. J. Padilla Ybarra y E. Romero A., «Metodología para realizar una automatización usando PLC,» *IMPULSO, REVISTA DE ELÉCTRONICA, ELÉCTRICA Y SISTEMAS COMPUTACIONALES,* vol. 1, nº 1, pp. 18-21, Diciembre 2005.

[2] M. N. Thomas C, «FPGA Implementation of a fastflex Supervisory Control and Data Acquisition,» *IEEE,* pp. 2216-2219, 2005.

[3] Maquiclick, «Ventajas y Desventajas de los PLC (Controlador Lógico Programable),» [En línea]. Available: https://www.fabricantes-maquinaria-industrial.es/ventajas-y-desventajas-de-los-plc-controlador-logico-programable/. [Último acceso: 29 mayo 2019].

[4] P. Paloma, «Observatorio Tecnológico,» Ministerio de Educación y Cultura de España, Octubre 2007. [En línea]. Available: http://recursostic.educacion.es/observatorio/web/gl/component/content/article/502-monografico-lenguajes-de-programacion?start=2.

[5] T. Technologies, «DE0-Nano Texas Instrument,» 2003-2013. [En línea]. Available: https://www.ti.com/lit/ug/tidu737/tidu737.pdf.

[6] T. Technologies, «AMAZON,» Terasic Technologies, [En línea]. Available: https://www.amazon.com/Terasic-Technologies-P0496-DE10-Nano-Kit/dp/B07B89YHSB/ref=sr_1_2?__mk_es_US=%C3%85M%C3%85%C5%BD%C3%95%C3%91&keywords=fpga+de0nano&qid=1559440898&s=gateway&sr=8-2.

[7] B. Muñoz Barrón, «Controlador modular y reconfigurable para máquina de inyección de plástico basado en FPGA,» *Universida Autónoma de Querétaro,* nº RI001279, pp. 3-14, Jun 2011.

[8] B. d. M. C. y. G. D. S. G. Sánchez Rivera, «Implementación en dispositivos FPGA de controladores para sistemas lineales inestables con retardo de tiempo.,» Congreso Nacional de Control Automatico, México DF, 2007.

[9] XILINX, «Fiel Programmable Gate Array (FPGA),» Xilinx Inc, [En línea]. Available: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html.

[10] E. Rojas, «¿QUÉ SON LOS FPGAS? ¿CÓMO FUNCIONAN? ¿PARA QUÉ SIRVEN? ¿QUIÉN DEBERÍA UTILIZARLOS?,» Nodo Electrónico, 1 Enero 2016. [En línea]. Available: https://nodoelectronico.com/2016/01/01/que-son-los-fpgas-como-funcionan-para-que-sirven-quien-deberia-utilizarlos/.

[11] Academy By Bit2me, «Qué es FPGA,» [En línea]. Available: https://academy.bit2me.com/que-es-fpga/.

[12] F. Pérez Bosch, «Aplicaciones de las FPGA,» GENERA TECNOLOGIAS, 08 Noviembre 2018. [En línea]. Available: https://www.generatecnologias.es/aplicaciones_fpga.html.

[13] E. y. C. Departamento de Ingenieria Electrica, «Controladores Lógicos Programables (PLC),» [En línea]. Available: http://www.ieec.uned.es/investigacion/Dipseil/PAC/archivos/Informacion_de_referencia_ISE6_1_1.pdf.

[14] INTEL, «FPFA Desing Software,» INTEL QUARTUS PRIME SOFWARE SUIT, [En línea]. Available: https://www.intel.la/content/www/xl/es/software/programmable/quartus-prime/overview.html.

[15] INTEL, «Differences Among Intel SoC Device,» [En línea]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/soc-fpga/uf_01005.pdf. [Último acceso: 18 Agosto 2019].

[16] terasIC, «DE1-SoC Board,» [En línea]. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=167&No=836&PartNo=2. [Último acceso: 20 Agosto 2019].

[17] teriasIC, «DE0-Nano-SoC Kit/Atlas-SoC Kit,» Design Service, [En línea]. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=167&No=941&PartNo=2. [Último acceso: 20 Octubre 2019].

[18] terasIC, «DE10-Nano Kit,» Desingn Service, [En línea]. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=167&No=1046&PartNo=4. [Último acceso: Agosto Agosto 2019].

[19] tercasIC, «DE10-Standard,» Desing Service, [En línea]. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=167&No=1081&PartNo=2. [Último acceso: 20 Agosto 2019].

[20] A. N. Dwi Cahyono, «Design and realization camera controller for a Remote Sensing Payload of nanosatellite FPGA (Field Programmable Gate Array) system based,» *IEEE,* nº 13708609, pp. 73-77, 2013.

[21] V. V. e. M. M. K. i. A. V. K. Milan I. Radulovi, «Realization of the teleprotection equipment interface for working with PLC equipment using FPGA logic,» *IEEE,* pp. 525-528, 2012.

[22] ElUniverso, *Las malas prácticas ambientales dañan el Río Daule,* 24 Septiembre 2017.

[23] «DE1-SoC Board,» Design Service, [En línea]. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=167&No=836&PartNo=2. [Último acceso: 20 Agosto 2019].

ANEXOS

Anexos 1: Programa en Cuartos

Anexos 1.1: Generador PWM

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.componentes.all;

entity pwm is
    port(
        Resetn, clock:  in std_logic;
        freqPWM : in std_logic_vector(8 downTo 0); --de 1 a 500KHz
        en  : in std_logic;
        ChargeFreq  :   in std_logic;
        dutyCycle   : in std_logic_vector(6 downTo 0);
        PWMSignal   : out std_logic;
        finCharge   : out Std_logic;
        clockPWM        : buffer std_logic
    );
end pwm;

Architecture sol of pwm is
    type estado is (S1, S2, S3);
    signal y: estado;
    signal LdA, ini, finDiv, vcc, gnd, ldC, men, may, igu: std_logic;
    signal men2, may2, igu2, pwms: std_logic;
    signal s50000: std_logic_vector(15 downTo 0);
    signal sFreqPwm, resto, nDiv: std_logic_vector(15 downTo 0);
    signal Acomp: std_logic_vector(8 downTo 0);
    signal Bcomp:   std_logic_vector(6 downTo 0);
    signal compmen1: std_logic_vector(8 downTo 0);
    signal pwmn: std_logic;
Begin
    -- transiciones
    Process(Resetn, clock)
    begin
        if Resetn = '0' then y <= S1;
        elsif (Clock'event and clock = '1') then
            case y is
                when S1 => if chargeFreq = '1' then y <= S2; end if;
                when S2 => y <= S3;
                when S3 => if finDiv = '1' then y<= S1; end if;
            end case;
        end if;
    end process;
    --salidas mss
    Process(Resetn, clock, finDIV)
    begin
        ldA <= '0'; ini <= '0'; finCharge <= '0';
        case y is
            when S1 =>
            when S2 => LDA <= '1';
```

```
            when S3 => ini <= '1';
                            if (finDIV = '1') then finCharge <= '1'; end
if;
        end case;
    end process;
    --Data processor
    s50000 <= "1100001101010000";
    compmen1 <= nDiv(8 downTo 0) - "000000001";
    multi100: mult100 generic map(n => 9)
        port map(freqPWM, sFreqPWM);
    Divisor: division generic map(n => 16)
        port map(clock, resetn, s50000, sFreqPWM, LdA, ini, findiv, resto,
nDiv);
    Contador: contador_up generic map(n => 9)
        port map(Resetn, clock, '1', LdC, "000000000", Acomp);
    Comp: comparador generic map(n => 9)
        port map(Acomp, compmen1, men, igu, may);
    LdC <= may or igu;
    camClock: ffJ port map(Resetn, clock, LdC, clockPWM);
    contDutyCycle: contador_up_0_99 port map(Resetn, clockPWM, '1', '0',
"0000000", BComp);
    compDutyCycle:  comparador generic map(n => 7)
        port map(Bcomp, dutyCycle, men2, igu2, may2);
    pwms <= men2;
    pwmn <= en and pwms;
    PWMsignal <= not pwmn;
end sol;
```

Anexos 1.2: soc_system.qsys

Use	Co...	Name	Description	Export	Clock	Base	End	IRQ
☑		hps_0	Arria V/Cyclone V Hard Processor System		*multiple*	0x0000_0000	0xffff_ffff	
☑		hps_only_ma...	JTAG to Avalon Master Bridge		clk_0			
☑		sysid_qsys	System ID Peripheral Intel FPGA IP		clk_0	0x0001_0000	0x0001_0007	
☑		onchip_memory...	On-Chip Memory (RAM or ROM) Intel FPGA IP		clk_0	0x0000_0000	0x0000_0fff	
☑		jtag_uart	JTAG UART Intel FPGA IP		clk_0	0x0002_0000	0x0002_0007	0
☑		fpga_only_ma...	JTAG to Avalon Master Bridge		clk_0			
☑		intr_capturer_0	Interrupt Capture Module		clk_0	0x0003_0000	0x0003_0007	
☑		clk_0	Clock Source		*exported*			
☑		PWMFreq	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_00d0	0x0000_00df	
☑		DutyCycle1	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_00c0	0x0000_00cf	
☑		DutyCycle2	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_00a0	0x0000_00af	
☑		DutyCycle3	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_0090	0x0000_009f	
☑		DutyCycle4	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_0080	0x0000_008f	
☑		DutyCycle5	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_0070	0x0000_007f	
☑		ControlPWM	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_00b0	0x0000_00bf	
☑		controlBomb	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_0020	0x0000_003f	
☑		Agitador	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_0060	0x0000_006f	
☑		ElectroValvula	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_0000	0x0000_001f	
☑		SensorNivel	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_0050	0x0000_005f	1
☑		Abosrbancia	PIO (Parallel I/O) Intel FPGA IP		clk_0	0x0000_0040	0x0000_004f	

Anexos 1.3

```
`define ENABLE_HPS
//`define ENABLE_CLK

module ghrd(

`ifdef ENABLE_CLK
      ///////// CLK /////////
      output              CLK_I2C_SCL,
      inout               CLK_I2C_SDA,
`endif /*ENABLE_CLK*/

      ///////// FPGA /////////
      input               FPGA_CLK1_50,
      input               FPGA_CLK2_50,
      input               FPGA_CLK3_50,
```

```
      ///////// GPIO /////////
      inout       [35:0] GPIO_0,
      input       [35:0] GPIO_1,

`ifdef ENABLE_HPS
      ///////// HPS /////////
      inout              HPS_CONV_USB_N,
      output      [14:0] HPS_DDR3_ADDR,
      output      [2:0]  HPS_DDR3_BA,
      output             HPS_DDR3_CAS_N,
      output             HPS_DDR3_CKE,
      output             HPS_DDR3_CK_N,
      output             HPS_DDR3_CK_P,
      output             HPS_DDR3_CS_N,
      output      [3:0]  HPS_DDR3_DM,
      inout       [31:0] HPS_DDR3_DQ,
      inout       [3:0]  HPS_DDR3_DQS_N,
      inout       [3:0]  HPS_DDR3_DQS_P,
      output             HPS_DDR3_ODT,
      output             HPS_DDR3_RAS_N,
      output             HPS_DDR3_RESET_N,
      input              HPS_DDR3_RZQ,
      output             HPS_DDR3_WE_N,
      output             HPS_ENET_GTX_CLK,
      inout              HPS_ENET_INT_N,
      output             HPS_ENET_MDC,
      inout              HPS_ENET_MDIO,
      input              HPS_ENET_RX_CLK,
      input       [3:0]  HPS_ENET_RX_DATA,
      input              HPS_ENET_RX_DV,
      output      [3:0]  HPS_ENET_TX_DATA,
      output             HPS_ENET_TX_EN,
      inout              HPS_GSENSOR_INT,
      inout              HPS_I2C0_SCLK,
      inout              HPS_I2C0_SDAT,
      inout              HPS_I2C1_SCLK,
      inout              HPS_I2C1_SDAT,
      inout              HPS_KEY,
      inout              HPS_LED,
      inout              HPS_LTC_GPIO,
      output             HPS_SD_CLK,
      inout              HPS_SD_CMD,
      inout       [3:0]  HPS_SD_DATA,
      output             HPS_SPIM_CLK,
      input              HPS_SPIM_MISO,
      output             HPS_SPIM_MOSI,
      inout              HPS_SPIM_SS,
      input              HPS_UART_RX,
      output             HPS_UART_TX,
      input              HPS_USB_CLKOUT,
      inout       [7:0]  HPS_USB_DATA,
      input              HPS_USB_DIR,
      input              HPS_USB_NXT,
      output             HPS_USB_STP,
`endif /*ENABLE_HPS*/
);


//=======================================================
//  REG/WIRE declarations
```

```
//=======================================================
// internal wires and registers declaration
  wire [1:0]  fpga_debounced_buttons;
  wire [7:0]  fpga_led_internal;
  wire        hps_fpga_reset_n;
  wire [2:0]  hps_reset_req;
  wire        hps_cold_reset;
  wire        hps_warm_reset;
  wire        hps_debug_reset;
  wire [27:0] stm_hw_events;

  //Asignacion de señales para el control de bombas
  wire [8:0]  pwmFreq;
  wire        ChargeFreq;
  wire [6:0]  DutyCycle1;
  wire [6:0]  DutyCycle2;
  wire [6:0]  DutyCycle3;
  wire [6:0]  DutyCycle4;
  wire [6:0]  DutyCycle5;
  wire        enBomb1;
  wire        enBomb2;
  wire        enBomb3;
  wire        enBomb4;
  wire        enBomb5;
  wire [4:0]  controlBomb;
  assign          controlBomb = {enBomb1, enBomb2, enBomb3, enBomb4,
enBomb5};

  wire        agitador;
  wire [14:0] electrovalvulas;


  assign      GPIO_0[10] = electrovalvulas[0];
  assign          GPIO_0[12] = electrovalvulas[1];
  assign          GPIO_0[14] = electrovalvulas[2];
  assign          GPIO_0[16] = electrovalvulas[3];
  assign          GPIO_0[18] = electrovalvulas[4];
  assign          GPIO_0[26] = electrovalvulas[5];
  assign          GPIO_0[24] = electrovalvulas[6];
  assign          GPIO_0[22] = electrovalvulas[7];
  assign          GPIO_0[20] = electrovalvulas[8];
  assign          GPIO_0[33] = electrovalvulas[9];
  assign          GPIO_0[35] = electrovalvulas[10];
  assign          GPIO_0[34] = electrovalvulas[11];
  assign          GPIO_0[32] = electrovalvulas[12];
  assign          GPIO_0[30] = electrovalvulas[13];
  assign          GPIO_0[28] = electrovalvulas[14];

  assign          GPIO_0[3] = agitador;



// connection of internal logics
  assign stm_hw_events    = {{13{1'b0}},SW, fpga_led_internal,
fpga_debounced_buttons};


//=======================================================
//  Structural coding
//=======================================================
```

```
soc_system u0 (
       //Clock&Reset
      .clk_clk(FPGA_CLK1_50 ),          //  clk.clk
      .reset_reset_n(1'b1          ),   //  reset.reset_n
      //HPS ddr3
      .memory_mem_a( HPS_DDR3_ADDR),    //  memory.mem_a
      .memory_mem_ba( HPS_DDR3_BA),     // .mem_ba
      .memory_mem_ck( HPS_DDR3_CK_P),   // .mem_ck
      .memory_mem_ck_n( HPS_DDR3_CK_N),     // .mem_ck_n
      .memory_mem_cke( HPS_DDR3_CKE),       // .mem_cke
      .memory_mem_cs_n( HPS_DDR3_CS_N),     // .mem_cs_n
      .memory_mem_ras_n( HPS_DDR3_RAS_N),   // .mem_ras_n
      .memory_mem_cas_n( HPS_DDR3_CAS_N),   // .mem_cas_n
      .memory_mem_we_n( HPS_DDR3_WE_N),     // .mem_we_n
      .memory_mem_reset_n( HPS_DDR3_RESET_N),      // .mem_reset_n
      .memory_mem_dq( HPS_DDR3_DQ),                // .mem_dq
      .memory_mem_dqs( HPS_DDR3_DQS_P),            // .mem_dqs
      .memory_mem_dqs_n( HPS_DDR3_DQS_N),          // .mem_dqs_n
      .memory_mem_odt( HPS_DDR3_ODT),              // .mem_odt
      .memory_mem_dm( HPS_DDR3_DM),                // .mem_dm
      .memory_oct_rzqin( HPS_DDR3_RZQ),            // .oct_rzqin
      //HPS ethernet
      .hps_0_hps_io_hps_io_emac1_inst_TX_CLK( HPS_ENET_GTX_CLK),
      .hps_0_hps_io_hps_io_emac1_inst_TXD0( HPS_ENET_TX_DATA[0] ),
      .hps_0_hps_io_hps_io_emac1_inst_TXD1    ( HPS_ENET_TX_DATA[1] ),
      .hps_0_hps_io_hps_io_emac1_inst_TXD2    ( HPS_ENET_TX_DATA[2] ),
      .hps_0_hps_io_hps_io_emac1_inst_TXD3    ( HPS_ENET_TX_DATA[3] ),
      .hps_0_hps_io_hps_io_emac1_inst_RXD0    ( HPS_ENET_RX_DATA[0] ),
      .hps_0_hps_io_hps_io_emac1_inst_MDIO    ( HPS_ENET_MDIO ),
      .hps_0_hps_io_hps_io_emac1_inst_MDC     ( HPS_ENET_MDC  ),
      .hps_0_hps_io_hps_io_emac1_inst_RX_CTL  ( HPS_ENET_RX_DV),
      .hps_0_hps_io_hps_io_emac1_inst_TX_CTL  ( HPS_ENET_TX_EN),
      .hps_0_hps_io_hps_io_emac1_inst_RX_CLK  ( HPS_ENET_RX_CLK),
      .hps_0_hps_io_hps_io_emac1_inst_RXD1    ( HPS_ENET_RX_DATA[1] ),
      .hps_0_hps_io_hps_io_emac1_inst_RXD2    ( HPS_ENET_RX_DATA[2] ),
      .hps_0_hps_io_hps_io_emac1_inst_RXD3    ( HPS_ENET_RX_DATA[3] ),
      //HPS SD card
      .hps_0_hps_io_hps_io_sdio_inst_CMD      ( HPS_SD_CMD    ),
      .hps_0_hps_io_hps_io_sdio_inst_D0       ( HPS_SD_DATA[0]     ),
      .hps_0_hps_io_hps_io_sdio_inst_D1       ( HPS_SD_DATA[1]     ),
      .hps_0_hps_io_hps_io_sdio_inst_CLK      ( HPS_SD_CLK   ),
      .hps_0_hps_io_hps_io_sdio_inst_D2       ( HPS_SD_DATA[2]     ),
      .hps_0_hps_io_hps_io_sdio_inst_D3       ( HPS_SD_DATA[3]     ),
      //HPS USB
      .hps_0_hps_io_hps_io_usb1_inst_D0       ( HPS_USB_DATA[0]    ),
      .hps_0_hps_io_hps_io_usb1_inst_D1       ( HPS_USB_DATA[1]    ),
      .hps_0_hps_io_hps_io_usb1_inst_D2       ( HPS_USB_DATA[2]    ),
      .hps_0_hps_io_hps_io_usb1_inst_D3       ( HPS_USB_DATA[3]    ),
      .hps_0_hps_io_hps_io_usb1_inst_D4       ( HPS_USB_DATA[4]    ),
      .hps_0_hps_io_hps_io_usb1_inst_D5       ( HPS_USB_DATA[5]    ),
      .hps_0_hps_io_hps_io_usb1_inst_D6       ( HPS_USB_DATA[6]    ),
      .hps_0_hps_io_hps_io_usb1_inst_D7       ( HPS_USB_DATA[7]    ),
      .hps_0_hps_io_hps_io_usb1_inst_CLK      ( HPS_USB_CLKOUT    ),
      .hps_0_hps_io_hps_io_usb1_inst_STP      ( HPS_USB_STP    ),
      .hps_0_hps_io_hps_io_usb1_inst_DIR      ( HPS_USB_DIR    ),
      .hps_0_hps_io_hps_io_usb1_inst_NXT      ( HPS_USB_NXT    ),
        //HPS SPI
      .hps_0_hps_io_hps_io_spim1_inst_CLK     ( HPS_SPIM_CLK  ),
      .hps_0_hps_io_hps_io_spim1_inst_MOSI    ( HPS_SPIM_MOSI ),
      .hps_0_hps_io_hps_io_spim1_inst_MISO    ( HPS_SPIM_MISO ),
```

```
      .hps_0_hps_io_hps_io_spim1_inst_SS0    ( HPS_SPIM_SS   ),
        //HPS UART
      .hps_0_hps_io_hps_io_uart0_inst_RX     ( HPS_UART_RX   ),
      .hps_0_hps_io_hps_io_uart0_inst_TX     ( HPS_UART_TX   ),
        //HPS I2C1
      .hps_0_hps_io_hps_io_i2c0_inst_SDA     ( HPS_I2C0_SDAT  ),
      .hps_0_hps_io_hps_io_i2c0_inst_SCL     ( HPS_I2C0_SCLK  ),
        //HPS I2C2
      .hps_0_hps_io_hps_io_i2c1_inst_SDA     ( HPS_I2C1_SDAT  ),
      .hps_0_hps_io_hps_io_i2c1_inst_SCL     ( HPS_I2C1_SCLK  ),
        //GPIO
      .hps_0_hps_io_hps_io_gpio_inst_GPIO09  ( HPS_CONV_USB_N ),
      .hps_0_hps_io_hps_io_gpio_inst_GPIO35  ( HPS_ENET_INT_N ),
      .hps_0_hps_io_hps_io_gpio_inst_GPIO40  ( HPS_LTC_GPIO   ),
      .hps_0_hps_io_hps_io_gpio_inst_GPIO53  ( HPS_LED   ),
      .hps_0_hps_io_hps_io_gpio_inst_GPIO54  ( HPS_KEY   ),
      .hps_0_hps_io_hps_io_gpio_inst_GPIO61  ( HPS_GSENSOR_INT ),

      .hps_0_f2h_stm_hw_events_stm_hwevents  (stm_hw_events),
      .hps_0_h2f_reset_reset_n               (hps_fpga_reset_n),
     .hps_0_f2h_warm_reset_req_reset_n      (~hps_warm_reset),
      .hps_0_f2h_debug_reset_req_reset_n     (~hps_debug_reset),
      .hps_0_f2h_cold_reset_req_reset_n      (~hps_cold_reset),
      .pwmfreq_export(pwmFreq),
      .controlpwm_export(ChargeFreq),
      .dutycycle1_export(DutyCycle1),   //dutycycle1.export
      .dutycycle2_export(DutyCycle2),   //dutycycle2.export
      .dutycycle3_export(DutyCycle3),   //dutycycle3.export
      .dutycycle4_export(DutyCycle4),   //dutycycle4.export
      .dutycycle5_export(DutyCycle5),   //dutycycle5.export
      .controlbomb_export(controlBomb), //controlbomb.export
      .agitador_export(agitador),       //agitador.export
      .electrovalvula_export(electrovalvulas),
      .sensornivel_export({GPIO_1[7],GPIO_1[5],GPIO_1[3],GPIO_1[1]}),
      .absorbancia_export
({GPIO_1[14],GPIO_1[12],GPIO_1[10],GPIO_1[8],GPIO_1[6],GPIO_1[4],GPIO_1[2],
GPIO_1[0]}
    );

 pwm    bomba1 (
    .Resetn(1'b1),
    .clock(FPGA_CLK1_50),
    .freqPWM(pwmFreq),
    .en(enBomb1),
    .ChargeFreq(ChargeFreq),
    .dutyCycle(DutyCycle1),
    .PWMSignal(GPIO_0[0])
 );

 pwm    bomba2 (
    .Resetn(1'b1),
    .clock(FPGA_CLK1_50),
    .freqPWM(pwmFreq),
    .en(enBomb2),
    .ChargeFreq(ChargeFreq),
    .dutyCycle(DutyCycle2),
    .PWMSignal(GPIO_0[2])
 );

  pwm   bomba3 (
    .Resetn(1'b1),
```

```
    .clock(FPGA_CLK1_50),
    .freqPWM(pwmFreq),
    .en(enBomb3),
    .ChargeFreq(ChargeFreq),
    .dutyCycle(DutyCycle3),
    .PWMSignal(GPIO_0[4])
 );

  pwm   bomba4 (
    .Resetn(1'b1),
    .clock(FPGA_CLK1_50),
    .freqPWM(pwmFreq),
    .en(enBomb4),
    .ChargeFreq(ChargeFreq),
    .dutyCycle(DutyCycle4),
    .PWMSignal(GPIO_0[6])
 );

  pwm   bomba5 (
    .Resetn(1'b1),
    .clock(FPGA_CLK1_50),
    .freqPWM(pwmFreq),
    .en(enBomb5),
    .ChargeFreq(ChargeFreq),
    .dutyCycle(DutyCycle5),
    .PWMSignal(GPIO_0[8])
 );
// Source/Probe megawizard instance
hps_reset hps_reset_inst (
  .source_clk (FPGA_CLK1_50),
  .source     (hps_reset_req)
);

altera_edge_detector pulse_cold_reset (
  .clk       (FPGA_CLK1_50),
  .rst_n     (hps_fpga_reset_n),
  .signal_in (hps_reset_req[0]),
  .pulse_out (hps_cold_reset)
);
  defparam pulse_cold_reset.PULSE_EXT = 6;
  defparam pulse_cold_reset.EDGE_TYPE = 1;
  defparam pulse_cold_reset.IGNORE_RST_WHILE_BUSY = 1;

altera_edge_detector pulse_warm_reset (
  .clk       (FPGA_CLK1_50),
  .rst_n     (hps_fpga_reset_n),
  .signal_in (hps_reset_req[1]),
  .pulse_out (hps_warm_reset)
);
  defparam pulse_warm_reset.PULSE_EXT = 2;
  defparam pulse_warm_reset.EDGE_TYPE = 1;
  defparam pulse_warm_reset.IGNORE_RST_WHILE_BUSY = 1;

altera_edge_detector pulse_debug_reset (
  .clk       (FPGA_CLK1_50),
  .rst_n     (hps_fpga_reset_n),
  .signal_in (hps_reset_req[2]),
  .pulse_out (hps_debug_reset)
);
  defparam pulse_debug_reset.PULSE_EXT = 32;
  defparam pulse_debug_reset.EDGE_TYPE = 1;
```

```
  defparam pulse_debug_reset.IGNORE_RST_WHILE_BUSY = 1;

endmodule
```

Anexos 2: Proyecto en C

Anexos 2.1: main.c

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <iostream>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "fpga.h"

#define HW_REGS_BASE ( ALT_STM_OFST )
#define HW_REGS_SPAN ( 0x04000000 )
#define HW_REGS_MASK ( HW_REGS_SPAN - 1 )

int tMuestra = 500000;
int tSulf = 500000;
short tAlivio = 30;
short valvAlivio1 = 11;
short valvAlivio2 = 12;
short valvAlivio3 = 13;
short valvMuestra = 1;
short valvSulf1 = 2;
short valvSulf2 = 3;
short valvSulf3 = 4;
short valvSulf5 = 5;
short bombMuestAgua = 0;
short bombSulf = 1;
int absorbancia;
float nitrito[3];
int i;
short solutoDisp = 0;
float media;
float varianza;
float desv;
int main() {
        printf("inicializando fpga \n");
        inicializarFPGA();
        printf("Carga Inicial de Soluto \n");
    cargaInicialSoluto();
        while(1){
        for(i = 0; i < 3; i++){ //Realiza el proceso 3 veces sobre el mismo
elemento
            printf("Extrayendo Dato %i de concentración \n", i);
            do{
                solutoDisp = 1;
                //Pregunta si hay soluciones disponibles
                if (nivelTank(0) == 1){
                    solutoDisp = 0;
                    printf("No hay suficiente muestra de agua para realizar
el proceso \n");
                }
                if (nivelTank(1) == 1){
                    solutoDisp = 0;
```

```
                    printf("No hay suficiente sulfanilamida para realizar
el proceso \n");
                }
                if (nivelTank(2) == 1){
                    solutoDisp = 0;
                    printf("No hay suficiente Naftiletilendiamina para
realizar el proceso \n");
                }
                if (nivelTank(3) == 1){
                    solutoDisp = 0;
                    printf("No hay suficiente Agua Destilada para realizar
el proceso \n");
                }
                if (solutoDisp == 0){
                    sleep(5);
                }
            }while(solutoDisp == 0);
            printf("1.- Inicio de toma de muestra de agua. \n");
            //Receso Inicial
                apagarActuadores();
            sleep(1);
            //Inicio
            obtener50mlMuestra();
            printf("2.- Iniciando toma de Sulfanilamida. \n");
            //Insertar Soluto 1
            obtener1mlSulfanilamida();
            printf("3.- Agitando mezcla por 1 minuto. Espere... \n");
            //agitar 1 min
            agitador(1);
//          sleep(60);
            agitador(0);
            printf("4.- Reposando mezcla por 5 minutos. Espere... \n");
//          sleep(300); //Espera 5 min para que la solucion repose
            //Insertar Soluto 2
            printf("5.- Iniciando toma de Naftiletilendiamina. \n");
            obtener1mlNaftiletilendiamina();
            //agitar 1 min
            printf("6.- Agitando mezcla por 1 minuto. Espere... \n");
            agitador(1);
//          sleep(60);
            agitador(0);
            printf("7.- Reposando mezcla por 10 minutos. Espere... \n");
//          sleep(600); //Reposo de la solución por 10 min
            //Pasar al colorimetro
            printf("8.- Enviando muestra al colorimetro. \n");
            activarValvula(14);
            activarPWM(4, 80, 5);
//          sleep(4); //espera a que se llene el colorimetro
            desactivarPWM(5);
            desactivarValvula(14);
            //Medir Dato Colorimetro
            absorbancia = getAbsorbancia();
            printf("9.- Abosrbancia obtenida: %i \n ", absorbancia/100);
            nitrito[i] = getConcentracion(absorbancia);
            printf("10.- Concentración (paso %i): %f \n", i+1, nitrito[i]);
            //Botar Desechos
            printf("11.- Enviando mezcla a Desechos \n");
            vertirDesechos();
            usleep(1000);
            printf("12.- Relizando limpieza \n");
            vertirAguaDestilada();
```

```
        }
        printf("Fin analisis de muestra \n");
        //Obtiene la media
        media = nitrito[0]+nitrito[1]+nitrito[2];
        media = media/3;
        //Oobtiene la varianza
        for(i = 0; i < 3; i++){
            nitrito[i] = nitrito[i]-media;
        }
        varianza = pow(nitrito[0], 2) + pow(nitrito[1], 2) +
pow(nitrito[2], 2);
        varianza = varianza/3;
        desv = sqrt(varianza);
        if(desv > 2){ //Si  la desviación es muy grande el dato es
incorrecto.
            printf("El resultado obtenido no es correco \n");
            printf("El dispositivo necesita revisión \n");
        }else{
            printf("Concentración de nitrito de la muestra: %f", media);
        }
        printf("FinProceso \n");
        }
    closeFPGA();
        return( 0 );

}
```

Anexos 2.2: fpga.h

```
#ifndef FPGA_H_   /* Include guard */
#define FPGA_H_

#include <stdio.h>
#include <stdlib.h>
#include <vector>
#include <time.h>
#include <inttypes.h>
#include <stdint.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <iostream>
#include <string.h>
#include <sstream>      // std::stringstream

#include "hwlib.h"
#include "socal/socal.h"
#include "socal/hps.h"
#include "socal/alt_gpio.h"
#include "hps_0.h"
using namespace std;


//Se especifican las funciones que usara main.c
int inicializarFPGA();
int closeFPGA();
void cambiarEstadoLED();
void activarPWM(short FreqPWM, short DutyCycle, short bomb);
void desactivarPWM(short bomb);
void activarValvula(short Valv);
void desactivarValvula(short Valv);
```

```
void agitador(short a);
int nivelTank(short a);
void apagarActuadores();
void obtener50mlMuestra();
void obtener1mlSulfanilamida();
void obtener1mlNaftiletilendiamina();
void vertirDesechos();
void vertirAguaDestilada();
void cargaInicialSoluto();
int getAbsorbancia();
float getConcentracion(int absorbancia);
#endif /* FPGA_H_ */
```

Anexos 2.3: fpga.c

```
//include de C
#include "fpga.h"
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <inttypes.h>
#include <stdint.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>

//include para FPGA
#include "hwlib.h"
#include "socal/socal.h"
#include "socal/hps.h"
#include "socal/alt_gpio.h"
#include "hps_0.h"

//definicion de registros base para la FPGA
#define HW_REGS_BASE ( ALT_STM_OFST )
#define HW_REGS_SPAN ( 0x04000000 )
#define HW_REGS_MASK ( HW_REGS_SPAN - 1 )
void *virtual_base;


volatile int *LED; //Instancia del Qsys como registro
volatile int *FREQPWM;
volatile int *DUTYCYCLE1;
volatile int *DUTYCYCLE2;
volatile int *DUTYCYCLE3;
volatile int *DUTYCYCLE4;
volatile int *DUTYCYCLE5;
volatile int *CONTROLPWM;
volatile int *ACTIVARBOMB;
volatile int *ELECTROVALV;
volatile int *AGITADOR;
volatile int *NIVEL;
volatile int *ABSORBANCIA;

int fd;


//Funciones Iniciales----------------------
```

```
int inicializarFPGA()
{
    if( ( fd = open( "/dev/mem", ( O_RDWR | O_SYNC ) ) ) == -1 ){
        printf( "ERROR: no se pudo abrir \"/dev/mem\"...\n" );
        return( 1 );
    }

    virtual_base = mmap( NULL, HW_REGS_SPAN, ( PROT_READ | PROT_WRITE ),
MAP_SHARED, fd, HW_REGS_BASE );

    if( virtual_base == MAP_FAILED ) {
        printf( "ERROR: mmap() fallo ... \n" );
        close( fd );
        return( 1 );
    }
    //Instancias de Qsys
    LED = (int*)(virtual_base + ( ( unsigned long  )( ALT_LWFPGASLVS_OFST +
LED_BASE ) & ( unsigned long)( HW_REGS_MASK ) )); //Cada instancia nueva
solo cambia LED_BASE con el nombre de la base de la isntancia
    FREQPWM  = (int*)(virtual_base + ( ( unsigned long  )(
ALT_LWFPGASLVS_OFST + PWMFREQ_BASE ) & ( unsigned long)( HW_REGS_MASK ) ));
    DUTYCYCLE1  = (int*)(virtual_base + ( ( unsigned long  )(
ALT_LWFPGASLVS_OFST + DUTYCYCLE1_BASE ) & ( unsigned long)( HW_REGS_MASK )
));
    DUTYCYCLE2  = (int*)(virtual_base + ( ( unsigned long  )(
ALT_LWFPGASLVS_OFST + DUTYCYCLE2_BASE ) & ( unsigned long)( HW_REGS_MASK )
));
    DUTYCYCLE3  = (int*)(virtual_base + ( ( unsigned long  )(
ALT_LWFPGASLVS_OFST + DUTYCYCLE3_BASE ) & ( unsigned long)( HW_REGS_MASK )
));
    DUTYCYCLE4  = (int*)(virtual_base + ( ( unsigned long  )(
ALT_LWFPGASLVS_OFST + DUTYCYCLE4_BASE ) & ( unsigned long)( HW_REGS_MASK )
));
    DUTYCYCLE5  = (int*)(virtual_base + ( ( unsigned long  )(
ALT_LWFPGASLVS_OFST + DUTYCYCLE5_BASE ) & ( unsigned long)( HW_REGS_MASK )
));
    CONTROLPWM  = (int*)(virtual_base + ( ( unsigned long  )(
ALT_LWFPGASLVS_OFST + CONTROLPWM_BASE ) & ( unsigned long)( HW_REGS_MASK )
));
    ACTIVARBOMB = (int*)(virtual_base + ( ( unsigned long  )(
ALT_LWFPGASLVS_OFST + CONTROLBOMB_BASE ) & ( unsigned long)( HW_REGS_MASK )
));
    ELECTROVALV = (int*)(virtual_base + ( ( unsigned long  )(
ALT_LWFPGASLVS_OFST + ELECTROVALVULA_BASE ) & ( unsigned long)(
HW_REGS_MASK ) ));
    AGITADOR = (int*)(virtual_base + ( ( unsigned long  )(
ALT_LWFPGASLVS_OFST + AGITADOR_BASE ) & ( unsigned long)( HW_REGS_MASK )
));
    NIVEL = (int*)(virtual_base + ( ( unsigned long  )( ALT_LWFPGASLVS_OFST
+ SENSORNIVEL_BASE ) & ( unsigned long)( HW_REGS_MASK ) ));
    ABSORBANCIA = (int*)(virtual_base + ( ( unsigned long  )(
ALT_LWFPGASLVS_OFST + ABOSRBANCIA_BASE ) & ( unsigned long)( HW_REGS_MASK )
));
}


int closeFPGA()
{
        if( munmap( virtual_base, HW_REGS_SPAN ) != 0 ) {
                printf( "ERROR: munmap() failed...\n" );
                close( fd );
```

```
                return( 1 );
        }

        close( fd );

}

//Funcion para LED ---------------

void cambiarEstadoLED(){
    *(LED) = *(LED)+0x01;  //Lee el registro de offset 0, que es donde esta
la DATA y suma 1 para cambiar el estado del LED.
    printf("Changed \n");
}

int nivelTank(short a){
    switch(a)
    {
    case 0:
        return (*NIVEL & 0x1);
    case 1:
        return (*NIVEL & 0x2);
    case 2:
        return (*NIVEL & 0x4);
    default:
        return (*NIVEL & 0x8);
        }
}

void agitador(short a){
    if (a == 0){
        *(AGITADOR) = 0x0;
    }
    else{
        *(AGITADOR) = 0x1;
    }
}

void activarValvula(short Valv){
    switch(Valv)
    {
    case 1:
        *(ELECTROVALV + 4) = 0x0001;
        break;
    case 2:
        *(ELECTROVALV + 4) = 0x0002;
        break;
    case 3:
        *(ELECTROVALV +4) = 0x0004;
        break;
    case 4:
        *(ELECTROVALV +4) = 0x0008;
        break;
    case 5:
        *(ELECTROVALV +4) = 0x0010;
                break;
    case 6:
        *(ELECTROVALV +4) = 0x0020;
                break;
    case 7:
        *(ELECTROVALV +4) = 0x0040;
```

```
                break;
    case 8:
        *(ELECTROVALV +4) = 0x0080;
                break;
    case 9:
        *(ELECTROVALV +4) = 0x0100;
                break;
    case 10:
        *(ELECTROVALV +4) = 0x0200;
                break;
    case 11:
        *(ELECTROVALV +4) = 0x0400;
                break;
    case 12:
        *(ELECTROVALV +4) = 0x0800;
                break;
    case 13:
        *(ELECTROVALV +4) = 0x1000;
                break;
    case 14:
        *(ELECTROVALV +4) = 0x2000;
                break;
    case 15:
        *(ELECTROVALV +4) = 0x4000;
                break;
    default:
        *(ELECTROVALV +4) = 0x8000;
                break;
    }
}


void desactivarValvula(short Valv){
        switch(Valv)
        {
        case 1:
                *(ELECTROVALV + 5) = 0x0001;
                break;
        case 2:
                *(ELECTROVALV + 5) = 0x0002;
                break;
        case 3:
                *(ELECTROVALV +5) = 0x0004;
                break;
        case 4:
                *(ELECTROVALV +5) = 0x0008;
                break;
        case 5:
                *(ELECTROVALV +5) = 0x0010;
                break;
        case 6:
                *(ELECTROVALV +5) = 0x0020;
                break;
        case 7:
                *(ELECTROVALV +5) = 0x0040;
                break;
        case 8:
                *(ELECTROVALV +5) = 0x0080;
                break;
        case 9:
                *(ELECTROVALV +5) = 0x0100;
```

```
        break;
    case 10:
                *(ELECTROVALV +5) = 0x0200;
                break;
        case 11:
                *(ELECTROVALV +5) = 0x0400;
                break;
        case 12:
                *(ELECTROVALV +5) = 0x0800;
                break;
        case 13:
                *(ELECTROVALV +5) = 0x1000;
                break;
        case 14:
                *(ELECTROVALV +5) = 0x2000;
                break;
        case 15:
                *(ELECTROVALV +5) = 0x4000;
                break;
        default:
                *(ELECTROVALV +5) = 0x8000;
                break;
        }
}

void activarPWM(short FreqPWM, short DutyCycle, short bomb){
    *(FREQPWM) = FreqPWM;
    *(CONTROLPWM) = 0x01;
        sleep(0.10);
        *(CONTROLPWM) = 0x00;
    switch(bomb)
    {
    case 1:
        *(DUTYCYCLE1) = DutyCycle;
        *(ACTIVARBOMB+4) = 0x10;
        break;
    case 2:
        *(DUTYCYCLE2) = DutyCycle;
                *(ACTIVARBOMB+4) = 0x08;
                break;
    case 3:
        *(DUTYCYCLE3) = DutyCycle;
                *(ACTIVARBOMB+4) = 0x04;
                break;
    case 4:
        *(DUTYCYCLE4) = DutyCycle;
                *(ACTIVARBOMB+4) = 0x02;
                break;
    default:
        *(DUTYCYCLE5) = DutyCycle;
                *(ACTIVARBOMB+4) = 0x01;
                break;
    }
}

void desactivarPWM(short bomb){
    switch(bomb)
        {
        case 1:
                *(ACTIVARBOMB+5) = 0x10;
                break;
```

```
        case 2:
                *(ACTIVARBOMB+5) = 0x08;
                break;
        case 3:
                *(ACTIVARBOMB+5) = 0x04;
                break;
        case 4:
                *(ACTIVARBOMB+5) = 0x02;
                break;
        default:
                *(ACTIVARBOMB+5) = 0x01;
                break;
        }

}

void apagarActuadores(){
    int i;
    for(i = 1; i < 6; i++){
        desactivarPWM(i);
    }
    for(i = 1; i < 16; i++){
        desactivarValvula(i);
    }
    agitador(0);
}

void obtener50mlMuestra(){
    printf("    Inicio toma de 50ml de muestra \n");
    short tActivacion = 50;
    short tVaciado = 5;
    activarValvula(12); //activa valvula 12; abre paso a la muestra
    //usleep(1000);
    activarPWM(4, 80, 4); //activa bomba 4
    sleep(tActivacion); //tiempo de activación de la bomba
    activarValvula(2); //activa valvula 2
    //usleep(1000);
    desactivarValvula(12); //Cierra la valvula 12; cierra paso a la muestra
    sleep(tVaciado); //espera a que el liquido de la manguera se vacie
    desactivarPWM(4); //detiene la bomba
    desactivarValvula(2); //cierra la valvula
    printf("    Muestra depositada en el tanque \n");
}

void obtener1mlSulfanilamida(){
    printf("    Inicio toma de 1ml de Sulfanilamida \n");
    short tCiclo = 30;
    short tRetorno = 25;
    short tAlivio = 20;
    //Ciclo de compuesto
    activarValvula(3);
    activarValvula(4);
    activarValvula(5);
    activarPWM(4, 100, 1);
    sleep(8);
    activarPWM(4, 40, 1);
    sleep(tCiclo); //Espera un tiempo para llenar el volumen deseado
    desactivarValvula(4);
    desactivarValvula(3);
    desactivarPWM(1);
    usleep(1000);
```

```
    //Regreso de sobrante
    activarValvula(2);
    activarPWM(4,100,1);
    sleep(tRetorno); //Espera que todo el liquido sobrante regrese al
recipiente
    desactivarPWM(1);
    desactivarValvula(2);
    desactivarValvula(5);
    usleep(1000);
    //Depositar en el tanque
    activarValvula(1);
    activarValvula(4);
    activarValvula(6);
    activarPWM(4, 100, 1);
    sleep(tAlivio); //Espera a que el millitro de sulfanilamida sea
depositado en el tanque
    desactivarPWM(1);
    desactivarValvula(1);
    desactivarValvula(4);
    desactivarValvula(6);
    printf("    sulfanilamida depositada en el tanque \n");
}

void obtener1mlNaftiletilendiamina(){
        printf("    Inicio toma de 1ml de Naftiletilendiamina \n");
        short tCiclo = 30;
        short tRetorno = 25;
        short tAlivio = 20;
        //Ciclo de compuesto
        activarValvula(7);
        activarValvula(8);
        activarValvula(9);
        activarPWM(4, 100, 2);
    sleep(8);
    activarPWM(4, 40, 2);
        sleep(tCiclo); //Espera un tiempo para llenar el volumen deseado
        desactivarValvula(8);
        desactivarValvula(7);
    desactivarPWM(2);
        usleep(1000);
        //Regreso de sobrante
        activarValvula(2);
        activarPWM(4,100,2);
        sleep(tRetorno); //Espera que todo el liquido sobrante regrese al
recipiente
        desactivarPWM(2);
        desactivarValvula(2);
        desactivarValvula(9);
        usleep(1000);
        //Depositar en el tanque
        activarValvula(1);
        activarValvula(8);
        activarValvula(10);
        activarPWM(4, 100, 2);
        sleep(tAlivio); //Espera a que el millitro de sulfanilamida sea
depositado en el tanque
        desactivarPWM(2);
        desactivarValvula(1);
        desactivarValvula(8);
        desactivarValvula(10);
        printf("    Naftiletilendiamina depositada en el tanque \n");
```

```
}

void vertirDesechos(){
    short tEspera = 25;
    printf("    Vertiendo Desechos \n");
    activarValvula(14);
    activarPWM(4, 100, 5);
    sleep(tEspera);
    desactivarPWM(5);
    desactivarValvula(14);
    printf("    Desechos Vertidos \n");
}

void vertirAguaDestilada(){
    short tEspera = 20;
    short tAlivio = 20;
    short tLimpiado = 20;
    printf("    Ingresando Agua Destilada \n");
    activarValvula(3);
    activarPWM(4, 80, 4);
    sleep(tEspera);
    desactivarValvula(3);
    activarValvula(2);
    sleep(tAlivio);
    desactivarPWM(4);
    desactivarValvula(2);
    printf("    Agitando \n");
    agitador(1);
    sleep(60);
    agitador(0);
    printf("    Limpiando \n");
    activarValvula(14);
    activarPWM(4, 100, 5);
    sleep(tLimpiado);
    desactivarPWM(5);
    desactivarValvula(14);
    printf("    Limpiado finalizado \n");
}

void cargaInicialSoluto(){
    activarValvula(3);
    activarValvula(4);
    activarValvula(5);
    activarValvula(7);
    activarValvula(8);
    activarValvula(9);
    activarPWM(4, 100, 1);
    activarPWM(4, 100, 2);
    sleep(10);
    apagarActuadores();
    usleep(1000);
    activarValvula(1);
    activarValvula(4);
    activarValvula(5);
    activarPWM(4, 100, 1);
    activarValvula(8);
    activarValvula(9);
    activarPWM(4, 100, 2);
    sleep(20);
    activarValvula(2);
    desactivarValvula(1);
```

```
    sleep(12);
    apagarActuadores();
}

int getAbsorbancia(){
    return (int)(*(ABSORBANCIA) & 0xff);
}

float getConcentracion(int absorbancia){
    float abs = absorbancia/100;
    float concentracion;
    concentracion = 1.8675*(abs);
    concentracion = concentracion - 0.228;
    return concentracion;
}
```

Anexos 2.4: Makefile

```
TARGET = main

CROSS_COMPILE = arm-linux-gnueabihf-
ALT_DEVICE_FAMILY ?= soc_cv_av
CFLAGS = -I /usr/include/ -I /usr/include/$(ALT_DEVICE_FAMILY)/ -D
$(ALT_DEVICE_FAMILY)
LDFLAGS = -g -Wall
CC = $(CROSS_COMPILE)g++
ARCH= arm

build: $(TARGET)

$(TARGET):
    g++ -g -Wall -std=gnu++11 $(LDFLAGS) $(CFLAGS) -c fpga.c -o fpga.o
    g++ -g -Wall -std=gnu++11 $(LDFLAGS) $(CFLAGS) main.c -o main.o fpga.o
```