Neiva, 11 de diciembre de 2017

Señores

CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN

UNIVERSIDAD SURCOLOMBIANA

Neiva, Huila

El (Los) suscrito(s):

JAIME HERNÁN BERMEO RAMÍREZ , con C.C. No. 1.075.293.224

JUAN DAVID OLMOS GARZÓN , con C.C. No. 1.075.277.137

_____, con C.C. No. _____,

_____, con C.C. No. _____,

autor(es) de la tesis y/o trabajo de grado o _____

titulado IMPLEMENTACIÓN DE UN PLC DE CÓDIGO ABIERTO CON IoT Y MONITOREO REMOTO

presentado y aprobado en el año 2017 como requisito para optar al título de

INGENIERO ELECTRÓNICO ;

Autorizo (amos) al CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN de la Universidad Surcolombiana para que, con fines académicos, muestre al país y el exterior la producción intelectual de la Universidad Surcolombiana, a través de la visibilidad de su contenido de la siguiente manera:

- Los usuarios puedan consultar el contenido de este trabajo de grado en los sitios web que administra la Universidad, en bases de datos, repositorio digital, catálogos y en otros sitios web, redes y sistemas de información nacionales e internacionales “open access” y en las redes de información con las cuales tenga convenio la Institución.

- Permita la consulta, la reproducción y préstamo a los usuarios interesados en el contenido de este trabajo, para todos los usos que tengan finalidad académica, ya sea en formato Cd-Rom o digital desde internet, intranet, etc., y en general para cualquier formato conocido o por conocer, dentro de los términos establecidos en la Ley 23 de 1982, Ley 44 de 1993, Decisión Andina 351 de 1993, Decreto 460 de 1995 y demás normas generales sobre la materia.

- Continúo conservando los correspondientes derechos sin modificación o restricción alguna; puesto que de acuerdo con la legislación colombiana aplicable, el presente es un acuerdo jurídico que en ningún caso conlleva la enajenación del derecho de autor y sus conexos.

De conformidad con lo establecido en el artículo 30 de la Ley 23 de 1982 y el artículo 11 de la Decisión Andina 351 de 1993, “Los derechos morales sobre el trabajo son propiedad de los autores”, los cuales son irrenunciables, imprescriptibles, inembargables e inalienables.

EL AUTOR/ESTUDIANTE:

Firma: _____

EL AUTOR/ESTUDIANTE:

Firma: _____

EL AUTOR/ESTUDIANTE:

Firma: _____

EL AUTOR/ESTUDIANTE:

Firma: _____

TÍTULO COMPLETO DEL TRABAJO: Implementación de un PLC de código abierto con IoT y monitoreo remoto.

AUTOR O AUTORES:

Primero y Segundo Apellido	Primero y Segundo Nombre
Bermeo Ramírez	Jaime Hernán
Olmos Garzón	Juan David

DIRECTOR Y CODIRECTOR TESIS:

Primero y Segundo Apellido	Primero y Segundo Nombre
Sendoya Losada	Diego Fernando

ASESOR (ES):

Primero y Segundo Apellido	Primero y Segundo Nombre
Robayo Betancourt	Faiber Ignacio
Soto Otalora	Agustín

PARA OPTAR AL TÍTULO DE: Ingeniero Electrónico

FACULTAD: Ingeniería

PROGRAMA O POSGRADO: Ingeniería Electrónica

CIUDAD: Neiva **AÑO DE PRESENTACIÓN:** 2017 **NÚMERO DE PÁGINAS:** 100

TIPO DE ILUSTRACIONES (Marcar con una **X**):

Diagramas **X** Fotografías **X** Grabaciones en discos___ Ilustraciones en general **X** Grabados___ Láminas___ Litografías___ Mapas___ Música impresa___ Planos **X** Retratos___ Sin ilustraciones___ Tablas o Cuadros **X**

SOFTWARE requerido y/o especializado para la lectura del documento: Lector pdf.

MATERIAL ANEXO:

PREMIO O DISTINCIÓN *(En caso de ser LAUREADAS o Meritoria)*:

PALABRAS CLAVES EN ESPAÑOL E INGLÉS:

Español	Inglés	Español	Inglés
1. PLC	PLC	6.	
2. OpenPLC	OpenPLC	7.	
3. IoT	IoT	8.	
4. Modbus	Modbus	9.	
5. Código abierto	Open source	10.	

RESUMEN DEL CONTENIDO: (Máximo 250 palabras)

En la actualidad la automatización de procesos industriales requiere un hardware y software privativo de alto costo, dificultando su adquisición para ambientes académicos y zonas de bajos recursos.

Este documento describe las etapas de desarrollo del prototipo de un PLC (Controlador Lógico Programable) basado en el proyecto de código abierto OpenPLC, con una arquitectura modular y extensible de bajo costo, comparable a los utilizados en la industria, capaz de ejecutar programas del estándar IEC 61131-3 y ser usado de manera segura en un entorno académico. Para demostrar esto se llevó a cabo una prueba de funcionalidad del prototipo frente a un PLC comercial, en la que se asignó la tarea de un control para el llenado y vaciado de dos tanques.

Adicionalmente, se realizaron algunas mejoras como integración y monitoreo remoto desde la nube, mediante la plataforma Firebase de Google, con un aplicativo móvil diseñado para tal fin.

ABSTRACT: (Máximo 250 palabras)

Nowadays, automatization of industrial processes requires high price privative hardware and software, making it's adquisition harder for academical environments and places which's population have a low income.

The main idea of this project was to develop a prototype of a PLC (Programmable Logic Controller), based on an open source project called OpenPLC, using a modular, extensible and low-price architecture, which is comparible to that used in the industry, capable of executing programs meeting the standard IEC 61131-3 ensuring it's use for a safe application on an academical environment.

Additionally, improvements were made as integration and remote monitoring from the cloud, using the platform Firebase from google, with a mobile application devised to that end.

APROBACION DE LA TESIS

Nombre Presidente Jurado: Diego Fernando Sendoya Losada

Firma: 

Nombre Jurado: Faiber Ignacio Robayo Betancourt

Firma: 

Nombre Jurado: Agustín Soto Otálora

Firma: 

IMPLEMENTACIÓN DE UN PLC DE CÓDIGO ABIERTO CON IOT Y MONITOREO REMOTO

JAIME HERNÁN BERMEO RAMÍREZ
JUAN DAVID OLMOS GARZÓN

UNIVERSIDAD SURCOLOMBIANA
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
NEIVA – HUILA
2017

IMPLEMENTACIÓN DE UN PLC DE CÓDIGO ABIERTO CON IOT Y MONITOREO REMOTO

JAIME HERNÁN BERMEO RAMÍREZ
JUAN DAVID OLMOS GARZÓN

Trabajo de grado presentado como requisito para optar al título de:
Ingeniero Electrónico

Director:
DIEGO FERNANDO SENDOYA LOSADA
Magíster en Ingeniería de Control Industrial

UNIVERSIDAD SURCOLOMBIANA
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
NEIVA – HUILA
2017

Nota de aceptación:

_____

_____

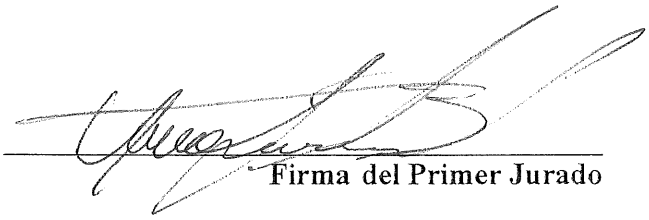
_____

_____

_____

_____

Firma del Director del Proyecto



Firma del Primer Jurado

Firma del Segundo Jurado

Neiva, 7 de Diciembre **de 2017**

(Dedicatoria)

A nuestros padres, hermanos y a la memoria de aquellos seres queridos que con su abnegada voluntad nos encaminaron por el sendero del triunfo.

AGRADECIMIENTOS

Agradecemos a la Universidad Surcolombiana, al programa de ingeniería electrónica y al cuerpo docente por su labor académica que contribuyo a nuestra formación durante todos estos años.

A nuestro director el ingeniero Diego Fernando Sendoya Losada, por su constante apoyo y guía desde el inicio.

A nuestra gestora del Tecnoparque del SENA nodo Neiva Hanny Callejas Reyes, a los ingenieros Carlos Flórez y Carlos Pérez, por su invaluable colaboración y seguimiento durante todo el desarrollo del proyecto.

A Thiago Alves y la comunidad de software libre, por el soporte brindado en las dificultades y dudas que se presentaron en cada etapa.

Finalmente, a nuestros compañeros y amigos, que con su granito de arena hicieron de todo este proceso algo más agradable.

RESUMEN

En la actualidad la automatización de procesos industriales requiere un hardware y software privativo de alto costo, dificultando su adquisición para ambientes académicos y zonas de bajos recursos.

Este documento describe las etapas de desarrollo del prototipo de un PLC (Controlador Lógico Programable) basado en el proyecto de código abierto OpenPLC, con una arquitectura modular y extensible de bajo costo, comparable a los utilizados en la industria, capaz de ejecutar programas del estándar IEC 61131-3 y ser usado de manera segura en un entorno académico. Para demostrar esto se llevó a cabo una prueba de funcionalidad del prototipo frente a un PLC comercial, en la que se asignó la tarea de un control para el llenado y vaciado de dos tanques.

Adicionalmente, se realizaron algunas mejoras como integración y monitoreo remoto desde la nube, mediante la plataforma Firebase de Google, con un aplicativo móvil diseñado para tal fin.

Palabras Clave: PLC, OpenPLC, IoT, Modbus, Código abierto.

ABSTRACT

Nowadays, automatization of industrial processes requires high price privative hardware and software, making it's adquisition harder for academical environments and places which's population have a low income.

This document describes the develop stages of the prototype of a PLC (Programmable Logic Controller) based on the OpenPLC open source project, using a modular, extensible and low-price architecture, which is comparible to that used in the industry, capable of executing programs meeting the standard IEC 61131-3 ensuring it's use for a safe application on an academical environment. To prove this, a performance test against a standard PLC was performed, which assigned a control task to fill and empty a couple of tanks.

Additionally, improvements were made as integration and remote monitoring from the cloud, using the platform Firebase from google, with a mobile application devised to that end.

Keywords: PLC, OpenPLC, IoT, Modbus, Open Source.

CONTENIDO

INTRODUCCIÓN..... 1

1. EL CONTROLADOR LÓGICO PROGRAMABLE..... 3

1.1 ARQUITECTURA INTERNA DEL PLC..... 3

1.1.1 Unidad Central de Proceso..... 3

1.1.2 Memoria del Autómata..... 4

1.1.3 Interfaces de Entrada y Salida..... 4

1.1.4 Fuente de Alimentación..... 5

1.2 ESTRUCTURA EXTERNA DEL PLC..... 6

1.3 PROTOCOLO DE COMUNICACIÓN..... 7

1.4 UNIDAD Y LENGUAJE DE PROGRAMACIÓN..... 9

2. DESARROLLO DEL SISTEMA OPENPI CONTROLLER..... 10

2.1 DISEÑO DEL HARDWARE..... 10

2.1.1 Unidad Central de Proceso..... 11

2.1.2 Diseño módulo de expansión entradas digitales..... 13

2.1.3 Diseño módulo de expansión salidas digitales..... 22

2.1.4 Diseño módulo de expansión entradas y salidas analógicas..... 29

2.1.4.1Entradas Analógicas..... 29

2.1.4.2 Salida Analógica..... 34

2.1.5 Fuente de Alimentación..... 36

2.1.6 Estructura Física..... 37

2.2 DISEÑO DEL SOFTWARE..... 39

2.2.1 Configuración e Instalación de la Raspberry Pi..... 39

2.2.2 Instalación de OpenPLC 43
2.2.3 Diseño de la Capa de Hardware de OpenPi Controller 46
2.2.4 Integración del OpenPi Controller en la Nube 51
2.2.5 Diseño de la Aplicación de Control y Supervisión Remota: OpenPi Watcher 58
2.2.5.1 Versión Móvil 58
2.2.5.2 Versión de Escritorio 61
3. PLCOPEN EDITOR: PROGRAMACIÓN DEL OPENPI CONTROLLER 63
3.1 DESCARGA E INSTALACIÓN 65
3.2 INSTRUCCIONES DE USO 66
4. RESULTADOS 71
5. CONCLUSIONES Y RECOMENDACIONES 76
5.1 CONCLUSIONES 76
5.2 RECOMENDACIONES 77
BIBLIOGRAFÍA 78
ANEXOS 79

LISTA DE FIGURAS

Figura 1. Diagrama de Bloques de un autómata programable 4

Figura 2. Tipos de Estructuras del PLC 6

Figura 3. Relación de red Maestro-Esclavo 7

Figura 4. Modelo de red del protocolo Modbus 8

Figura 5. Tipos de lenguajes de programación 9

Figura 6. Componentes de Hardware OpenPi Controller 10

Figura 7. Raspberry Pi 2 Modelo B 11

Figura 8. Pinout Raspberry Pi 13

Figura 9. Diagrama de bloques de una entrada discreta 14

Figura 10. Diagrama electrónico de una entrada digital 15

Figura 11. Expansor remoto de E/S de 8 bits PCF8574 15

Figura 12. Diagrama de bloques de una salida discreta 23

Figura 13. Diagrama electrónico de una salida digital 24

Figura 14. Conversor analógico/digital ADS1115 29

Figura 15. Diagrama de bloques funcional del ADS1115 30

Figura 16. Diagrama electrónico de una entrada analógica 31

Figura 17. Señal de Salida Filtro pasa-bajo 32

Figura 18. Señal de entrada analógica filtrada y rectificada 33

Figura 19. Convertidor digital-analógico MCP4725 34

Figura 20. Diagrama de bloques funcional del MCP4725 34

Figura 21. Diagrama electrónico de la interfaz de Salida Analógica de 0 a 10 V 36

Figura 22. Fuente de alimentación conmutada AC-DC 24V/6A 36

Figura 23. Estructura externa del OpenPi Controller 38

Figura 24. Identificación de los archivos CAD del prototipo OpenPi Controller 39

Figura 25. Página de descargas de RPI 40

Figura 26. Versiones de Descarga de NOOBS 40

Figura 27. Herramientas para Formateo de SD 41

Figura 28. Selección de Sistema Operativo a instalar 41

Figura 29. Progreso de Instalación del Sistema Operativo NOOBS 42

Figura 30. Entorno de Escritorio 42

Figura 31. Paquetes adicionales 43

Figura 32. Clon de Repositorio Oficial 44

Figura 33. Proceso de Compilación de OpenPLC 44

Figura 34. Selección de Capa de Hardware 45

Figura 35. Interfaz Web de OpenPLC 45

Figura 36. Interacción de los componentes de software y hardware del PLC 46

Figura 37. Diagrama de Rutina Principal de OpenPLC 46

Figura 38. Diagrama del proceso de Inicialización del Hardware de OpenPi Controller 48

Figura 39. Diagrama de actualización de Buffers de OpenPi Controller 50

Figura 40. Sitio oficial de Firebase 51

Figura 41. Consola de Firebase 52

Figura 42. Base de Datos en tiempo real de OpenPi Controller 52

Figura 43. Políticas de Acceso a la base de datos 53

Figura 44. Diagrama de enlace de OpenPi Controller con Firebase 54

Figura 45. Ítem de configuración de Firebase 55

Figura 46. Clave API del proyecto Picontroller en Firebase 55

Figura 47. Diagrama de Clase ModbusStream 56

Figura 48. Diagrama de Detección de Cambio de estados del OpenPi Controller 57

Figura 49. Diagrama de operación de la aplicación OpenPi Watcher 58

Figura 50. Pestaña Coils 59

Figura 51. Pestaña Digital Inputs 60

Figura 52. Pestaña Holding Registers 60

Figura 53. Pestaña Input Registers 61

Figura 54. Pestañas de la Aplicación versión escritorio 62

Figura 55. Modelo de datos de PLCopen Editor 63

Figura 56. Lenguajes de Programación PLCopen Editor 64

Figura 57. Descarga PLCopen Editor 65

Figura 58. Ejecución PLCopen Editor 66

Figura 59. Ventana Principal de PLCopen Editor 67

Figura 60. Elementos de diseño de Diagrama Ladder PLCopen Editor 68

Figura 61. Carga y Compilación de un Programa al OpenPi Controller 70

Figura 62. Modelo a escala de un sistema de tanques 71

Figura 63. PLC Micro850 Allen Bradley 72

Figura 64. Diagrama Ladder en PLCopen Editor 72

Figura 65. Diagrama Ladder en Connected Workbench Components 73

Figura 66. Tabla de variables y direcciones - Workbench 73

Figura 67. Tabla de variables y direcciones – PLCOpen Editor 73

Figura 68. Estados de salidas digitales OpenPi Controller vs Micro850 74

LISTA DE TABLAS

Tabla 1. Tipos y Funciones de las interfaces de E/S 5

Tabla 2. Bloques de Modelo de Datos de Modbus 8

Tabla 3. Características de la RPI 2 Modelo B 12

Tabla 4. Características eléctricas del PCF8574 16

Tabla 5. Direccionamiento I2C del PCF8574 16

Tabla 6. Selecciones de filtro rápido de entrada discreta 19

Tabla 7. Direccionamiento I2C del ADS1115 30

Tabla 8. Características eléctricas del ADS1115 31

Tabla 9. Características eléctricas del MCP4725 35

Tabla 10. Direccionamiento I2C del MCP4725 35

Tabla 11. Características eléctricas fuente de alimentación 37

Tabla 12. Funciones y Códigos de Operación de Modbus TCP 59

Tabla 13. Dirección de Registros OpenPi Controller 69

GLOSARIO

Aislamiento Galvánico: Consiste en la separación de partes funcionales de un circuito eléctrico para prevenir el traspaso de portadores de carga.

API: Interfaz de programación de aplicaciones, representa la capacidad de comunicación entre componentes de *software*.

Búfer: Memoria de almacenamiento temporal de información que permite transferir los datos entre unidades funcionales con características de transferencia diferentes.

Clase: En programación, es una plantilla para la creación de objetos de datos según un modelo predefinido. Cada clase es un modelo que define un conjunto de variables -el estado, y métodos apropiados para operar con dichos datos -el comportamiento.

CTR: Razón de Transferencia de Corriente, es un parámetro fundamental de los optoacopladores, se refiere a la proporción del valor de la corriente de salida a la corriente de entrada

Framework: Conocido como entorno de trabajo, es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

Gestor de Paquetes: Colección de herramientas que sirven para automatizar el proceso de instalación, actualización, configuración y eliminación de paquetes de software.

GPIO: Entrada/Salida de Propósito General, es un pin genérico en un chip, cuyo comportamiento (incluyendo si es un pin de entrada o salida) se puede controlar (programar) por el usuario en tiempo de ejecución.

HMI: Interfaz de usuario por sus siglas en idioma inglés, (Human (Y) Machine Interface) que se usa para referirse a la interacción entre humanos y máquinas; Aplicable a sistemas de Automatización de procesos.

IoT: Internet de las cosas, es un concepto que se refiere a la interconexión digital de objetos cotidianos con internet

Objeto: Es una unidad dentro de un programa de computadora que consta de un estado y de un comportamiento, que a su vez constan respectivamente de datos almacenados y de tareas realizables durante el tiempo de ejecución.

Open Source: (*Código abierto*) es el término con el que se conoce al software distribuido y desarrollado libremente. El código abierto tiene un punto de vista más orientado a los beneficios prácticos de compartir el código que a las cuestiones éticas y morales las cuales destacan en el llamado software libre.

Optoelectrónica: La optoelectrónica es el nexo de unión entre los sistemas ópticos y los sistemas electrónicos. Los componentes optoelectrónicos son aquellos cuyo funcionamiento está relacionado directamente con la luz.

OSI: El modelo de interconexión de sistemas abiertos (ISO/IEC 7498-1), más conocido como "modelo OSI", (*Open System Interconnection*) es un modelo de referencia para los protocolos de la red de arquitectura en capas.

Periférico: Es la denominación genérica para designar al aparato o dispositivo auxiliar e independiente conectado a la unidad central de procesamiento de una computadora.

Python: Es un lenguaje de programación abierto y de alto nivel; con característica multiparadigma, es decir orientada a objetos, programación interpretativa y funcional.

SSH: (*Secure Shell*, en español: intérprete de órdenes seguro) es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder servidores privados a través de una puerta trasera (también llamada *backend*). Permite manejar por completo el servidor mediante un intérprete de comandos.

Streaming: Es la distribución digital de contenido multimedia a través de una red de computadoras, de manera que el usuario utiliza el producto a la vez que se descarga. La palabra retransmisión se refiere a una corriente continua que fluye sin interrupción, y habitualmente a la difusión de audio o vídeo.

INTRODUCCIÓN

Actualmente el control de procesos industriales exige sistemas de automatización con elevado grado de confiabilidad y disponibilidad, una operación clara y objetiva además de ser equipos de fácil mantenimiento. El controlador lógico programable o comúnmente nombrado PLC nace como una alternativa para controlar procesos en tiempo real y en ambientes agresivos, siendo el resultado de la evolución de aquellos equipos de control rígido diseñados para realizar exclusivamente una actividad específica de control correspondiente a un determinado proceso[1].

A pesar de que los autómatas programables han supuesto la aplicación masiva del microprocesador al mundo de los controles industriales; el uso de un PLC no sigue estando al alcance de todo el mundo, puesto que son sistemas de alto costo, herméticos en su funcionamiento, poco seguros y sobre todo, sus licencias propietarias restringen la posibilidad de modificación y el entendimiento de su operación interna en el ámbito académico, limitándose al de una caja negra con entradas y salidas definidas. Este escenario y otras muchas situaciones han sido responsables de limitar las posibilidades de un nuevo avance en el conocimiento de la humanidad, capaz de romper el enfoque tradicional del software privativo. Por esta razón, surge este proyecto como respuesta a la necesidad de adoptar las ventajas que conlleva el uso de software y hardware libre para el diseño de nuevas herramientas autómatas programables que estén al alcance de todas las personas y puedan ser empleadas en ambientes amigables y seguros para el aprendizaje del control de procesos industriales.

El presente trabajo tiene como objetivo principal la implementación de un prototipo de PLC de código abierto mediante una plataforma de bajo costo como la Raspberry Pi; gracias a su facilidad de integración con IoT contara con un aplicativo móvil que podrá supervisar y modificar el estado del proceso que se desee controlar y eliminar la necesidad de cableado para la programación del dispositivo. Este prototipo no pretende competir con un PLC comercial, sino por el contrario, contribuir al desarrollo de la materia electiva PLC en la Universidad Surcolombiana con la intención de que sea más fácil entender las características y funcionamiento de los mismos; se trata de un sistema que se va a usar solamente con propósitos educativos.

[1] MEDINA D. y ALVAREZ E. Controladores Lógicos Programables. Departamento de Electrónica y Control. Universidad del Zulia. República Bolivariana

Este documento está distribuido en cinco capítulos que describen el trabajo realizado para alcanzar el diseño final del prototipo. En el capítulo 1 se explican en forma general los componentes que conforman un controlador lógico programable. El capítulo 2 describe la implementación de la plataforma *Hardware/Software* desarrollada, denominada OpenPi Controller, partiendo de las adaptaciones de hardware realizadas para el diseño de los diferentes módulos e interfaces de entrada/salida y las adaptaciones de software para el control de la capa de hardware e intercambio de datos con otros dispositivos utilizando protocolo Modbus sobre TCP/IP. En el capítulo 3 se expone la estructura y funcionamiento del editor empleado para la programación del prototipo de PLC, el capítulo 4 resume los resultados de las pruebas de funcionalidad frente a un controlador de gama baja. Finalmente, en el capítulo 5 se presentan las conclusiones derivadas del trabajo desarrollado y el trabajo a futuro.

1. EL CONTROLADOR LÓGICO PROGRAMABLE

Un controlador lógico programable conocido como PLC es un aparato electrónico construido a partir de un microprocesador o microcontrolador que se utiliza en la automatización de procesos industriales. Este tipo de dispositivos poseen una configuración interna independiente y cuentan con una memoria capaz de almacenar programas escritos por el usuario, a través de un software que define la secuencia de operaciones o instrucciones necesarias para manejar una gran cantidad de equipos mediante las unidades de entrada y salida[2].

A diferencia de las computadoras de propósito general, el PLC está diseñado para múltiples señales de entrada y de salida, rangos de temperatura ampliados, inmunidad al ruido eléctrico y resistencia a la vibración y al impacto.

1.1 ARQUITECTURA INTERNA DEL PLC

El autómata se configura alrededor de una unidad central la cual se une por medio de buses internos a las diferentes interfaces de entrada y salida, y de memoria. Esta configuración es conocida como la arquitectura ***interna*** del autómata[3].

El PLC está compuesto esencialmente por los siguientes bloques funcionales, los cuales se pueden apreciar en la figura 1:

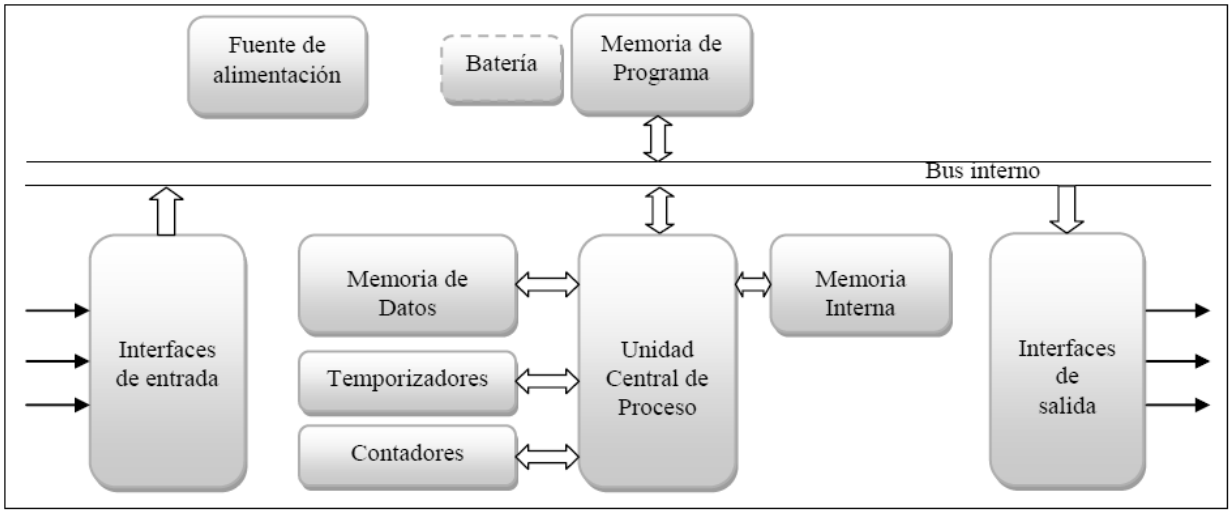
- Unidad central de proceso o de control, CPU.
- Memoria del autómata.
- Interfaces de entrada y salida.
- Fuente de alimentación

1.1.1 Unidad Central de Proceso

Se denomina CPU (Central Processing Unit) por sus siglas en ingles. Este bloque es el encargado de ejecutar el programa de usuario y de ordenar la transferencia de información en el sistema de entradas y salidas. También es el encargado de establecer la comunicación con periféricos externos, como lo son la unidad de programación, monitores y otros autómatas o computadoras.

[2] PORRAS A. y MONTANERO A.P., (1992). Autómatas Programables: fundamentos, manejo, instalación y prácticas.
[3] BALCELLS Josep y ROMERAL José Luis, (2000). Autómatas Programables. Barcelona.

Figura 1. *Diagrama de Bloques de un autómata programable*



Fuente: Imagen tomada y editada de Balcells, J. "Autómatas Programables"

1.1.2 Memoria del Autómata

La memoria de trabajo es donde el autómata guarda todo lo que necesita para ejecutar la tarea de control, esta generalmente se encuentra contenida en el CPU. Se pueden diferenciar dos tipos de memoria. La memoria de ejecución que contiene los programas permanentes, es no volátil por estar implementada con ROM y la memoria de aplicación en la cual se almacenan los programas del usuario y los datos que están cambiando, es implementada con memoria volátil RAM.

1.1.3 Interfaces de Entrada y Salida[4]

Las interfaces de entrada y salida establecen la comunicación entre la CPU y el proceso, filtrando, adaptando y codificando de forma comprensible para dicha unidad las señales procedentes de los elementos de entrada, y decodificando y amplificando las señales generales durante la ejecución del programa antes de enviarlas a los elementos de salida. Estas interfaces pueden clasificarse de distintas formas según se muestra en la tabla 1:

[4] BALCELLS Josep y ROMERAL José Luis, (2000). Autómatas Programables. Barcelona.

Tabla 1. *Tipos y Funciones de las interfaces de E/S*

Tipos	Codificación	Sentido	Funciones de la Interfaz
Todo o Nada	Binaria 1 bit	Entradas	• Adaptación de niveles de tensión • Filtrado de perturbaciones • Aislamiento galvánico
		Salidas	• Adaptación de niveles de tensión • Amplificación de corriente • Aislamiento galvánico
Señales Continuas	Analógicas (0, ± 10 V) (4, 20 mA)	Entradas	• Adaptación y filtrado de señal • Conversión A/D
		Salidas	• Conversión D/A • Adaptación a 0, ± 10 V o 4. 20 mA
	Digitales (8, 16… bits)	Entradas	• Selección de canal y multiplexado • Conversión de códigos
		Salidas	• Conversión de código (Bin. ↔ASCII 7 ↔ segmentos) • Amplificación de corriente
		Bidireccionales	• Conversión de código (serie ↔ paralelo) • Protocolo de diálogo (hard +soft)

Fuente: Tomada de Balcells, J. "Autómatas Programables".

1.1.4 Fuente de Alimentación

Las fuentes de alimentación proporcionan las tensiones necesarias para el funcionamiento de los distintos circuitos del sistema. Un autómata programable está formado por bloques que requieren niveles de tensión y de potencia diferentes y que, además, están sometidos a condiciones ambientales de ruido electromagnético también distintas.

La alimentación a la CPU puede ser de continua a 24 Vcc o en alterna a 110/220 Vca, en cualquier caso, la propia CPU alimenta las interfaces conectadas a través del bus interno. Por otro lado, la alimentación a los circuitos E/S puede realizarse según los distintos tipos en alterna a 48/110/220 Vca o en continua a 12/24/48 Vcc[5].

[5] BALCELLS Josep y ROMERAL José Luis, (2000). Autómatas Programables. Barcelona.

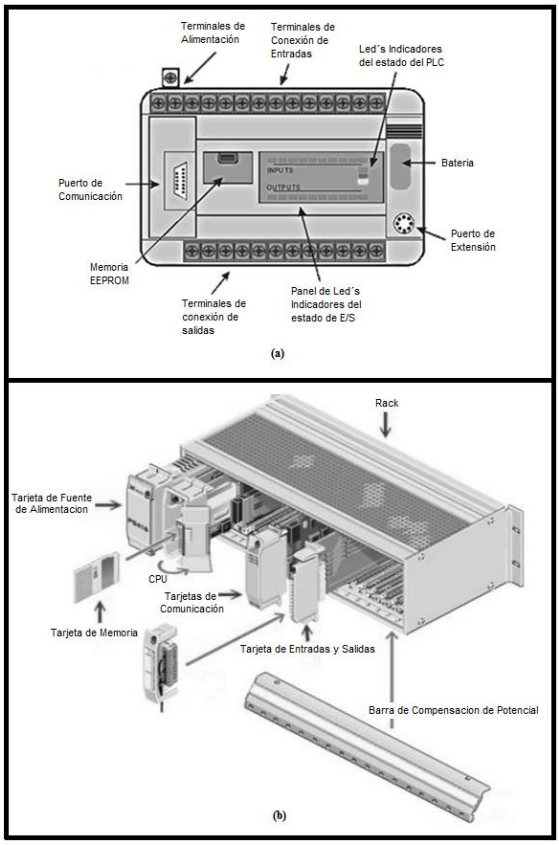
1.2 ESTRUCTURA EXTERNA DEL PLC[6]

El término estructura externa o configuración externa de un PLC industrial se refiere al aspecto físico exterior del mismo, bloques o elementos en que está dividido. Actualmente son tres las estructuras más significativas que existen en el mercado:

- ***Estructura compacta:*** Presenta en un solo bloque todos sus elementos, son los PLC de gama baja o nano-autómatas los que tienen esta estructura.
- ***Estructura semimodular:*** Separa las E/S del resto de los elementos, es la estructura típica de los PLC de gama media.
- ***Estructura modular:*** Existe un módulo para cada uno de los diferentes elementos que componen el PLC, los de gama alta son los que suelen tener una estructura modular.

En la figura 2 se pueden apreciar los dos tipos de estructuras principales.

Figura 2. *Tipos de Estructuras del PLC*



(a) Estructura del PLC Compacto. (b) Estructura del PLC Modular.
Fuente: http://saint-hyoga.wixsite.com/controleselectricos/en-blanco-cn5k

[6] PORRAS A. y MONTANERO A.P., (1992). Autómatas Programables: fundamentos, manejo, instalación y prácticas.

1.3 PROTOCOLO DE COMUNICACIÓN

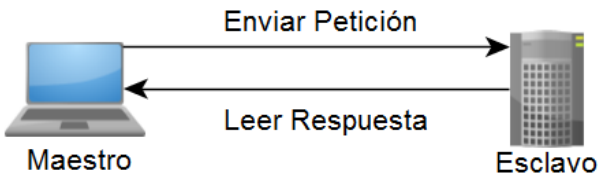
Los protocolos de comunicación para los controladores lógicos programables se basan en un sistema de transmisión de información por un sólo cable (*bus de campo*) que simplifica enormemente la instalación y operación de máquinas y equipamientos industriales utilizados en el proceso de producción. Debido a la falta de estándares, las compañías han desarrollado varias soluciones cada una de ellas con diferentes prestaciones y campos de aplicación[7].

En el ámbito de los autómatas cabe resaltar como fundamental sistema de transmisión de datos aquellos que están basados en una comunicación *maestro-esclavo* según el esquema *pregunta-respuesta*; pero de entre todos estos el que más cabe destacar es el protocolo MODBUS.

El Protocolo Modbus

Modbus es un protocolo de solicitud-respuesta implementado usando una relación maestro-esclavo. En una relación maestro-esclavo la comunicación siempre se produce en pares, un dispositivo debe iniciar una solicitud y luego esperar una respuesta y el dispositivo de inicio (el maestro) es responsable de iniciar cada interacción, como se ve en la figura 3.

Figura 3. *Relación de red Maestro-Esclavo*

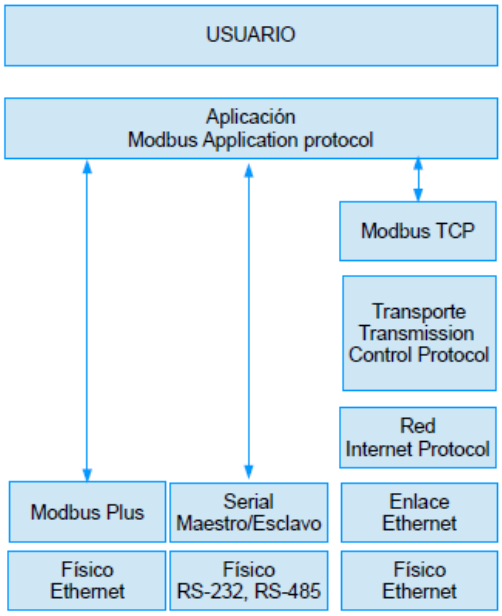


Fuente: http://www.ni.com/white-paper/52134/es/#toc1

Modbus está definido de acuerdo con el modelo OSI (Ver figura 4), está posicionado en la capa 7 de Aplicación del mismo, provee comunicación cliente- servidor entre los dispositivos conectados en diferentes tipos de buses o redes.

[7] SALAZAR C. y CORREA L. (2011). Buses de campo y Protocolos en redes industriales. Facultad de Ciencias e Ingeniería, Universidad de Manizales

Figura 4. *Modelo de red del protocolo Modbus*



Fuente: ModbusOrg (2006a). Modbus application protocol specification v1.1b. Technical report, The Modbus Organization.

El protocolo Modbus define el método con el que un PLC obtiene acceso a otro PLC, como responde un PLC a otros dispositivos y la detección y reporte de errores. Con el fin de aprovechar las herramientas ofrecidas por la internet, se desarrolló Modbus/TCP, el cual también está basado en el modelo OSI, aunque no utiliza todas las capas del modelo[8].

Los datos disponibles por medio de Modbus son almacenados en uno de los cuatro bancos de datos: *bobinas*, *entradas discretas*, *registros de retención* y *registros de entrada*. Estos definen el tipo y los derechos de acceso de los datos contenidos como se describe en la tabla 2.

Tabla 2. *Bloques de Modelo de Datos de Modbus*

Bloque de Memoria	Tipo de Datos	Acceso de Maestro	Acceso de Esclavo
Bobinas	Booleano	Lectura/Escritura	Lectura/Escritura
Entradas Discretas	Booleano	Solo Lectura	Lectura/Escritura
Registros de Retención	Palabra Sin Signo	Lectura/Escritura	Lectura/Escritura
Registros de Entrada	Palabra Sin Signo	Solo Lectura	Lectura/Escritura

Fuente: http://www.ni.com/white-paper/52134/es/#toc1

[8] ModbusOrg (2006a). Modbus application protocol specification v1.1b. Technical report, The Modbus Organization

1.4 UNIDAD Y LENGUAJE DE PROGRAMACIÓN

La unidad de programación es un aparato generalmente externo que se conecta al PLC cuando se desea modificar o transferir programas. Dependiendo del tipo de PLC se podrá usar un programador manual, una terminal o una computadora personal.

El Lenguaje de Programación en cambio, permite al usuario ingresar un programa de control como un conjunto de instrucciones en la memoria del PLC, usando una sintaxis establecida.

Hoy por hoy cada fabricante diseña su propio software de programación, lo que significa que existe una gran variedad comparable con la cantidad de controladores que hay en el mercado. No obstante, actualmente existen cuatro tipos de lenguajes de programación como los más difundidos a nivel mundial; estos se pueden agrupar o dividir en dos grupos.

Textuales

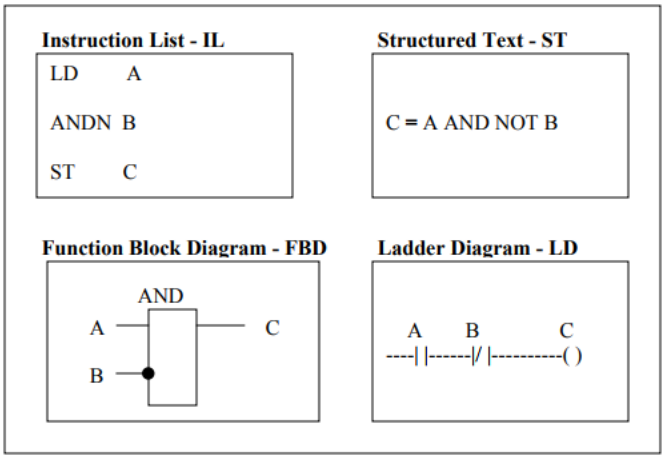
- Lista de Instrucciones (Instruction List – IL)
- Texto estructurado (Structured Tex – ST)

Gráficos

- Diagrama de Escalera (Ladder Diagram – LD)
- Diagrama de Bloques de Funciones (Function Block Diagram – FBD)

En la figura 5, se ilustra como los cuatro lenguajes describen la misma parte simple de un programa.

Figura 5. *Tipos de lenguajes de programación*



Fuente: Tomada de PLCopen TC1: Standards links to IEC 61131-3.
Disponible en http://www.plcopen.org/pages/tc1_standards/downloads/plcopen_iec61131-3_feb2014.pptx

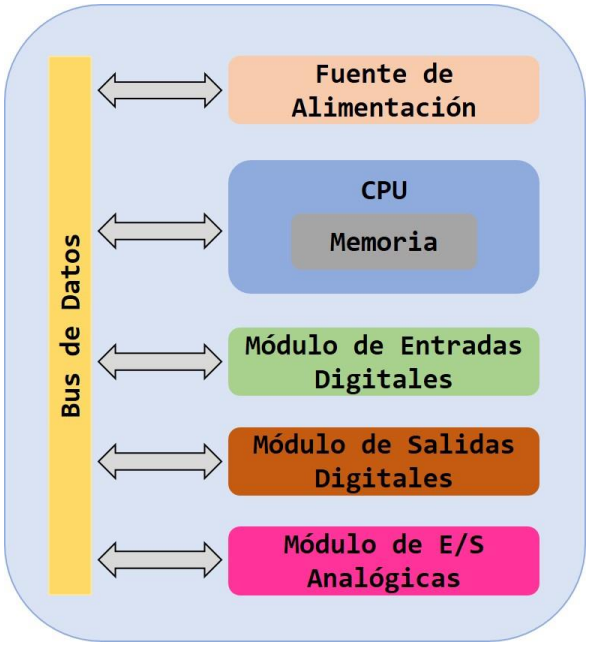
2. DESARROLLO DEL SISTEMA OPENPI CONTROLLER

El prototipo de PLC implementado “OpenPi Controller” se propuso a partir de la arquitectura estándar de un controlador lógico programable, se diseñó según una estructura modular y extensible de muy bajo costo. Su tamaño y otras características hacen que se asemeje a los típicos dispositivos de gama media/baja.

A continuación, se describen los componentes de hardware (Ver figura 6) que lo conforman (CPU, memoria, interfaces y fuente de alimentación) y las adaptaciones de software hechas para manejar los diferentes módulos de entrada y salida; además del diseño de la aplicación de supervisión y control del dispositivo.

2.1 DISEÑO DEL HARDWARE

Figura 6. *Componentes de Hardware OpenPi Controller*

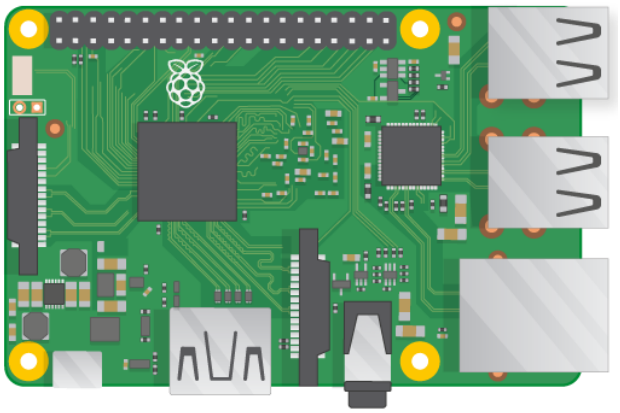


Fuente: Elaboración Propia

2.1.1 Unidad Central de Proceso

El papel de la CPU del prototipo se basó en la placa Raspberry Pi 2 modelo B (Ver figura 7). Se eligió esta plataforma de hardware libre y bajo costo por ser nativamente compatible con OpenPLC[9], el proyecto de código abierto creado por el ingeniero eléctrico *Thiago Alves*; sobre el que se apoyó la implementación y diseño de este trabajo, ya que permite emular el comportamiento de un PLC en diferentes plataformas tales como Arduino, Raspberry Pi, ESP8266, entre otras.

Figura 7. *Raspberry Pi 2 Modelo B*



Fuente: https://www.raspberrypi.org/learning/hardware-guide/components/raspberry-pi/

Esta plataforma aparte de ser compatible con el sistema OpenPLC se escogió por ser una computadora de placa simple que, a diferencia del Arduino tiene facilidades de integración con IoT que le permiten realizar tareas más avanzadas como el monitoreo e interconexión con la nube, sin necesidad de módulos externos. También es el hardware ideal para comunicarse por I2C a circuitos integrados periféricos y su sistema operativo se actualiza constantemente reduciendo las posibilidades de una brecha de seguridad a la hora de transmitir datos. En la tabla 3 se resumen algunas de las características principales de esta placa.

[9] T. R. Alves, M. Buratto, F. M. de Souza and T. V. Rodrigues, "OpenPLC: An open source alternative to automation" IEEE Global Humanitarian Technology Conference (GHTC 2014), San Jose, CA, 2014, pp. 585-589.

Tabla 3. *Características de la RPI 2 Modelo B*

Raspberry Pi 2 Modelo B	
SoC	BroadCom BCM2836
CPU	ARM11 ARMv7 – 900Mhz
GPU	BroadCom VideoCore IV 250 MHz OpenGL ES 2.0
RAM	1 GB LPDDR SDRAM 450 MHz
USB 2.0	4
Video	HDMI 1.4 @ 1920x1200 píxeles
Memoria	Micro-SD
Ethernet	10/100 Mbps
Tamaño	85.60 x 56.5 mm
Voltaje de Alimentación	5V, 2A

Fuente: https://www.raspberrypi.org/learning/hardware-guide/components/raspberry-pi/

La comunicación de la CPU con los demás módulos del PLC se realizó mediante la interfaz I2C, bus que facilita la comunicación entre microcontroladores, memorias y otros dispositivos, con solo dos líneas de señal y un común o masa. Su velocidad de transmisión es de unos 100 Kbits por segundo, aunque hay casos especiales en los que el reloj llega hasta los 3,4 MHz.

Descripción de las señales:

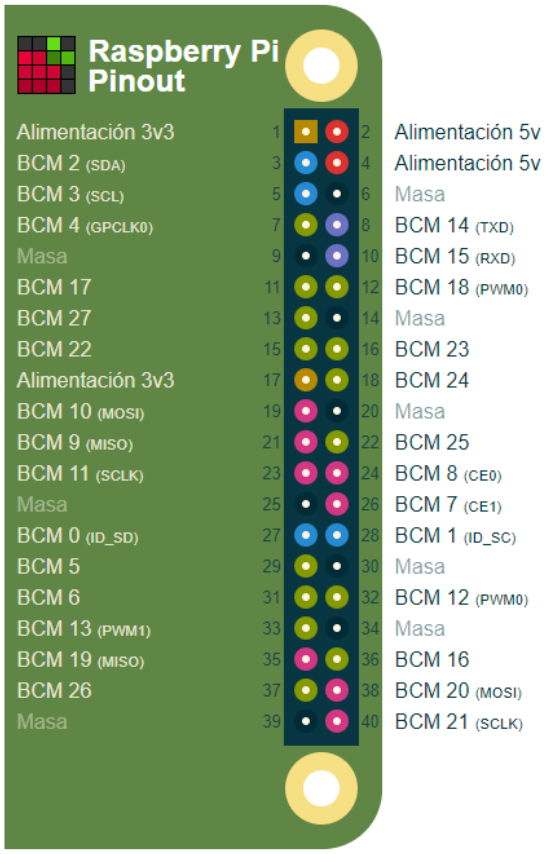
- SCL (System Clock) es la línea de los pulsos de reloj que sincronizan el sistema.
- SDA (System Data) es la línea por la que se mueven los datos entre los dispositivos.
- GND (Masa) común de la interconexión entre todos los dispositivos "enganchados" al bus.

La metodología de comunicación de datos del bus I2C es en serie y sincrónica; una de las señales del bus marca el tiempo (pulsos de reloj) y la otra se utiliza para intercambiar datos. Ya que pueden existir varios dispositivos conectados al bus, la comunicación se basa en un sistema de maestro-esclavo[10]. Para este caso el maestro será la CPU y los esclavos los demás módulos.

En la figura 8 se localizan los pines correspondientes a la interfaz I2C de la Raspberry Pi, que corresponde a los pines físicos 3 y 5, pero según la numeración de pin Broadcom (GPIO) serán los pines 2 (SDA) y 3 (SCL).

[10] The I2C Bus Specification (version 2.1, January 2000) Disponible en: http://www.semiconductors.philips.com/acrobat/literature/9398/39340011.pdf

Figura 8. *Pinout Raspberry Pi*



Fuente: https://es.pinout.xyz/pinout/

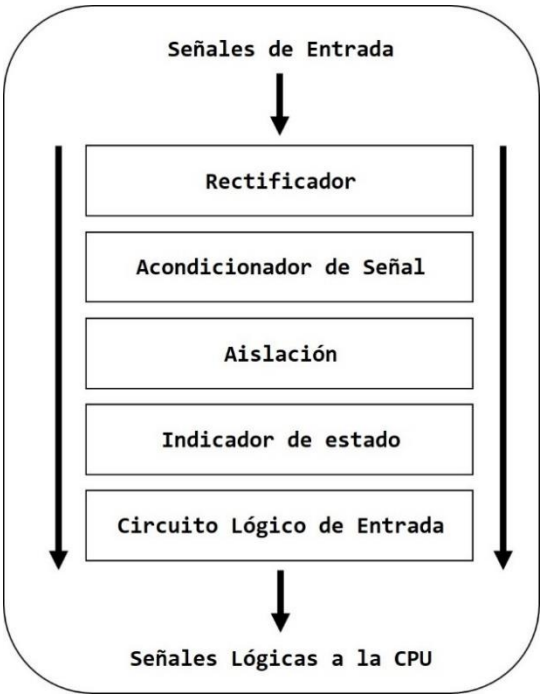
Este módulo en los archivos de diseño del proyecto será nombrado CORE o núcleo y cumplirá la labor de procesador y memoria del PLC. Es decir, ejecutara el programa de aplicación escrito por el usuario y administrara las tareas de comunicación entre dispositivos.

2.1.2 Diseño módulo de expansión entradas digitales

El módulo de entradas digitales será el que permitirá conectar al PLC captadores de señales discretas o binarias, las cuales solo pueden tomar dos estados lógicos (ON/OFF). Este módulo trabajara únicamente con señales de tensión de 24 VDC, que son las más comúnmente utilizadas y dispondrá de un total de 8 entradas digitales.

La estructura general sobre la que se basó el diseño de cada una de las entradas digitales puede separarse en varios bloques, como se muestra en la figura 9, por donde pasará la señal hasta convertirse en un 0 o un 1 para la CPU.

Figura 9. *Diagrama de bloques de una entrada discreta*



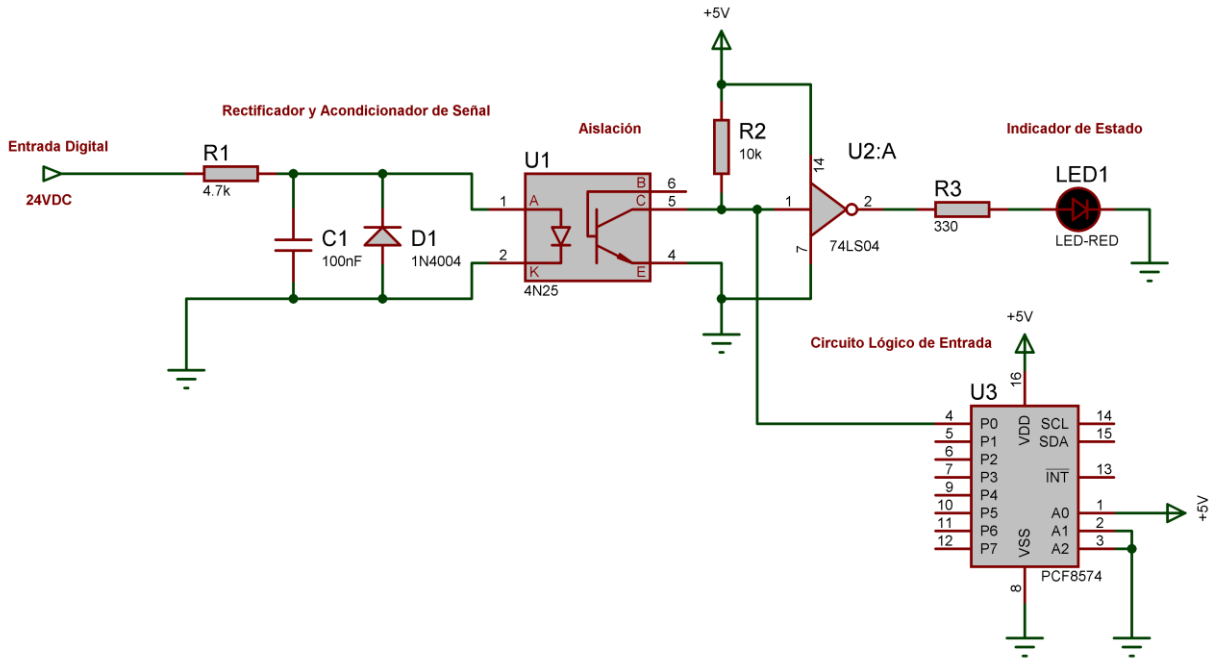
Fuente: Elaboración Propia

Estos bloques respectivamente son:

- ***Rectificador:*** Limita o impide daños por inversión de polaridad.
- ***Acondicionador de señal:*** Elimina ruidos eléctricos y detecta los niveles de señal para los que conmuta el estado lógico.
- ***Aislación:*** Separa las entradas del resto del PLC, evitando daños por sobretensión.
- ***Indicador de estados:*** Indica la presencia o ausencia de tensión en la entrada, se dispone de un indicador luminoso por canal.
- ***Circuito lógico de entrada:*** Es el encargado de informar a la CPU el estado de la entrada cuando ésta la interrogue.

A continuación, se explicará el diseño del circuito de una entrada digital del prototipo; en la figura 10 se observa el diagrama electrónico correspondiente.

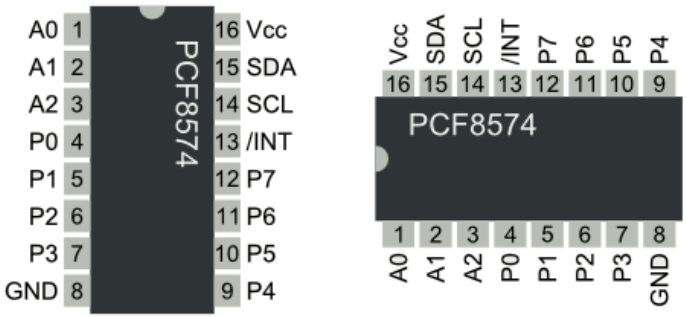
Figura 10. *Diagrama electrónico de una entrada digital*



Fuente: Elaboración Propia

Para poder manejar 8 entradas digitales se utilizó el integrado PCF8574, un expansor de E/S que consiste en un puerto cuasi bidireccional de 8 bits, con una interfaz I2C (Ver figura 11). Este dispositivo proporciona una expansión de uso remoto para la mayoría de las familias de microcontroladores a través de este bus de dos líneas, su puerto cuenta con salidas bloqueadas con capacidad de alta velocidad de corriente para conducir directamente leds. En la tabla 4 se presentan las principales características eléctricas del PCF8574.

Figura 11. *Expansor remoto de E/S de 8 bits PCF8574*



Fuente: http://www.ti.com/product/PCF8574

Tabla 4. *Características eléctricas del PCF8574*

Características Eléctricas	
Voltaje de Alimentación	2.5V a 6.0V
Voltaje de entrada	[-0.5 (min.), Vcc+0.5 (máx.)]V
Corriente de Entrada	±20mA
Corriente de Salida	±25mA
Potencia	400mW
Frecuencia de Operación	100 KHz
Rango de temperatura de operación	-40 a +85°C

Fuente: http://www.ti.com/lit/ds/symlink/pcf8574.pdf

Este integrado permite que cada E/S cuasi-bidireccional pueda utilizarse como una entrada o salida sin el uso de una señal de control de dirección de datos, para este primer módulo digital se configuro solo como puerto de entrada y se seleccionó la dirección de esclavo 0x21 para la comunicación con la CPU mediante el bus I2C; según el diagrama de bloques de la figura 9 este dispositivo actuaría como el ***circuito lógico de entrada***, manejando una lógica de voltaje TTL. En la tabla 5 se observan las conexiones respectivas de los pines A0, A1 y A2 a masa o nivel bajo (L) y VCC o nivel alto (H), para el direccionamiento del PCF8574 como dispositivo esclavo.

Tabla 5. *Direccionamiento I2C del PCF8574*

Inputs			I2C-BUS SLAVE ADDRESS
A2	**A1**	**A0**	
L	L	L	32 (decimal), 20 (hexadecimal)
L	L	H	33 (decimal), 21 (hexadecimal)
L	H	L	34 (decimal), 22 (hexadecimal)
L	H	H	35 (decimal), 23 (hexadecimal)
H	L	L	36 (decimal), 24 (hexadecimal)
H	L	H	37 (decimal), 25 (hexadecimal)
H	H	L	38 (decimal), 26 (hexadecimal)
H	H	H	39 (decimal), 27 (hexadecimal)

Fuente: http://www.ti.com/lit/ds/symlink/pcf8574.pdf

Para diseñar la interfaz de entrada que se encargara de recibir las señales de 24VDC y luego interactuar con el PCF8574, se partió por implementar en primer lugar la etapa de ***rectificación y acondicionamiento de la señal***; esta etapa se encuentra compuesta por un filtro pasa bajo y por un diodo 1N4004 que se utilizara como protección contra inversiones de polaridad. La siguiente etapa o etapa de ***aislación*** corresponde a un optoacoplador, también llamado aislador acoplado ópticamente, este es un dispositivo de emisión y recepción que funciona como un interruptor activado mediante la luz emitida por un diodo LED que satura un componente optoelectrónico, normalmente en forma de fototransistor o fototriac; para este caso se utilizó el integrado de referencia 4N25, un optoacoplador con una salida de tipo fototransistor. Este elemento tendrá como función acoplar la tensión de entrada (24V) al nivel manejado por el circuito lógico (5V).

Como se observa en la figura 10 la resistencia R1 se encargará de limitar la corriente que pasará por el diodo de luz del optoacoplador y junto con el condensador C1 formar un filtro RC pasa bajo para eliminar el ruido o las altas frecuencias; por otro lado, el diodo D1 protegerá al circuito del flujo inverso del voltaje de entrada y la resistencia R2 hará trabajar al fototransistor en modo conmutación. Finalmente, para la etapa del ***indicador de estado*** debido a que la lógica de los pines de entrada del PCF8574 es negada, se implementó una compuerta negadora 74LS04, junto a una resistencia R3 para cambiar el estado de la señal de salida del 4N25 (señal circuito lógico de entrada) y activar el LED1, que se utiliza para indicar la condición o estado de la entrada (activa/desactiva).

A continuación, se calcularán los valores respectivos de R1, C1, R2 y R3 para que el circuito de entrada funcione correctamente. Previamente fue necesario conocer algunos de los datos técnicos de un módulo de entradas digitales estandarizado, por tal motivo se utilizó como referencia los datos de dos módulos diferentes (*MicroLogix 1200* y *S7-1200*)[11], ambos de tipo modular. De la comparación entre ambos se obtuvieron los siguientes datos tomados exclusivamente para estos cálculos.

$$Condiciones\ de\ Entrada\colon\quad V_{entrada} = 24V \quad V_{IHmin} = 15V \quad V_{ILmax} = 5V$$
$$I_{Imax} = 4mA \quad I_{IHmax} = 2{,}5mA \quad I_{ILmax} = 1mA$$

[11] Controlador programable S7-1200. Manual de sistema. Siemens
Controladores programables MicroLogix™ 1200. Manual de Usuario. Allen-Bradley

De la misma forma fueron necesarios los siguientes datos técnicos del optoacoplador 4N25, los cuales se encuentran disponibles en su respectiva ficha técnica o de datos[12].

$$4N25: \quad CTR_{min} = 0{,}2 \quad V_{Fmin} = 1{,}2V \quad V_{Fmax} = 1{,}5V$$
$$I_{Fmax} = 50mA \quad I_{CEOmax} = 50nA \quad V_{CE(sat)max} = 0{,}5V$$

a) Elección de R1 y C1

El valor de R1 determinará la intensidad de corriente que circulará por el led (I_F) cuando este conduce, para ello I_F deberá satisfacer las siguientes condiciones:

$$I_F < I_{Fmax} = 50mA$$
$$I_F > I_{Imax} = 4mA$$

La primera condición viene impuesta por el Led del optoacoplador y la segunda por la capacidad de consumo de corriente máxima de una entrada digital típica (I_{Imax}). La condición más restrictiva es ésta última. La intensidad I_F se calcula fácilmente mediante el siguiente análisis.

$$I_F = \frac{V_{entrada} - V_F - 0}{R1}$$

Aplicando la condición más restrictiva a la ecuación anterior se obtiene el límite inferior para R1.

$$I_F > I_{Imax}$$
$$\frac{V_{entrada} - V_{Fmin} - 0}{R1} > I_{Imax}$$
$$R1 < \frac{V_{entrada} - V_{Fmin} - 0}{I_{Imax}} = \frac{24 - 1{,}2 - 0}{4mA} = 5{,}7k\Omega$$

Pero antes se debe tener en cuenta que esta resistencia R1 también formará parte del circuito RC, el cual establecerá la máxima frecuencia de operación permitida a la entrada. Para este diseño se optó por una frecuencia de corte de 0.375 kHz, que garantice un retardo de activación y desactivación de la entrada aproximadamente de 1.5 ms. Estas condiciones se muestran en la tabla 6, de donde se seleccionó el promedio de los valores centrales.

[12] Disponible en la web: http://www.vishay.com/docs/81864/4n25x000.pdf

Tabla 6. *Selecciones de filtro rápido de entrada discreta*

Retardo activación/ desactivación (ms)	**Máxima Frecuencia de Corte (Hz)**
0.500	1000
1.000	500
2.000	250
4.000	125

Fuente: Controladores programables MicroLogix™ 1200. Manual de Usuario. Allen-Bradley

Teniendo en cuenta la fórmula del filtro RC, se asumió el valor del capacitor C1 a 100nF y se calculó el valor respectivo de R1.

$$F_C = \frac{1}{2\pi R1 C1}$$

$$375\ Hz = \frac{1}{2\pi R1 \cdot 100 \times 10^{-9} F}$$

$$R1 = \frac{1}{2\pi \cdot 375 Hz \cdot 100 \times 10^{-9} F} \approx 4{,}3k\Omega$$

Siendo el valor comercial más cercano 4.7kΩ; aun con este valor de R1 se logran cumplir las condiciones necesarias para que el diodo led del optoacoplador emita la luz necesaria para la activar el fototransistor.

De acuerdo al circuito de la figura 10, la señal de entrada se acoplará ópticamente a la entrada de una compuerta 74LS04, a través del optoacoplador 4N25. Para asegurar el estado de conmutación del fototransistor fue necesario considerar tanto los datos técnicos del optoacoplador descritos previamente, como los de la compuerta negadora especificados a continuación; y así calcular el valor de R2 que permita al fototransistor actuar como un interruptor.

$V_{Colector} = 5V \pm 5\%$

$74LS04$: $I_{IHmax} = 20\mu A$ $\quad I_{ILmax} = -0{,}4mA$ $\quad I_{OLmax} = 8mA$

$V_{IHmin} = 2{,}0V$ $\quad V_{ILmax} = 0{,}8V$ $\quad V_{OLmax} = 0{,}5V$

$I_{OHmax} = -0{,}4mA$ $\quad V_{OHmin} = 2{,}7V$

Cuando un transistor se utiliza como interruptor o switch, la corriente de base debe tener un valor para lograr que el transistor entre en corte y otro para que entre en saturación. Esto dos casos se utilizarán para determinar el valor de R2 bajo las siguientes condiciones:

b) Fototransistor saturado

Cuando el fototransistor está saturado, la tensión a su salida ($V_{CE(sat)}$) debe ser interpretada como nivel bajo por el inversor:

$$\left(V_{CE(sat)max} = 0{,}5V\right) < (V_{ILmax} = 0{,}8V)$$

Para que el fototransistor este saturado:

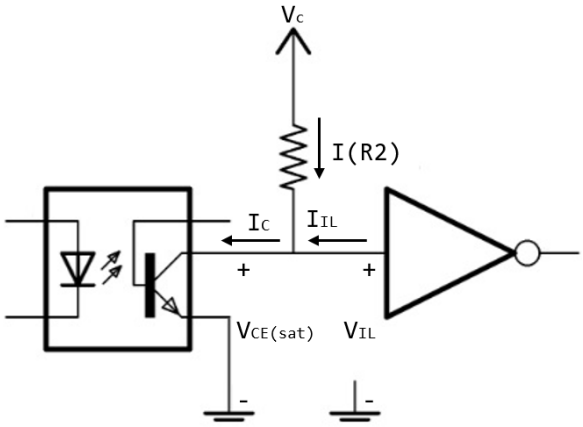
$$I_C < CTR\,.\,I_F = 0{,}2 \cdot I_F$$

A medida que R2 disminuye, I_C aumenta y el fototransistor puede llegar a salir de saturación; por tanto, existe una R2 mínima.

Para que la condición anterior se satisfaga siempre, imponemos que se satisfaga para el peor caso:

$$I_{C(\max)} < CTR\,.\;I_{F(\min)} = 0{,}2 \cdot I_{F(\min)}$$

La corriente I_C depende directamente de la resistencia R2, como se observa en el siguiente análisis.



$$I_C = I(R2) + I_{IL}$$

$$I_{C(max)} = \frac{V_{C(max)} - V_{CE(min)}}{R2} + I_{ILmax}$$

A menor resistencia R2, mayor corriente. Por otra parte, el valor mínimo de I_F se puede calcular de la siguiente manera:

$$I_{F(min)} = \frac{V_{entrada} - V_{Fmax} - 0}{R1} = \frac{24 - 1{,}5 - 0}{4{,}7 \times 10^3} = 4{,}8mA$$

Imponemos la condición de saturación del fototransistor:

$$\frac{V_{C(max)} - V_{CE(min)}}{R2} + I_{ILmax} < 0{,}2 \cdot I_{F(\min)}$$

Y despejamos el valor mínimo de R2:

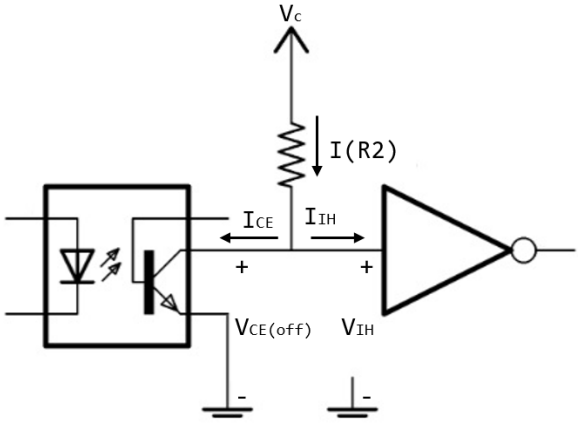
$$R2 > \frac{V_{C(max)} - V_{CE(min)}}{0{,}2 \cdot I_{F(\min)} - I_{ILmax}} = \frac{5{,}25 - 0}{(0{,}2 \cdot 4{,}8 \times 10^{-3}) - (-0{,}4 \times 10^{-3})} = 3{,}9k\Omega$$

c) Fototransistor en corte

Cuando el fototransistor está en corte, la tensión $V_{CE(off)}$ debe ser interpretada como un valor alto por el inversor y además respetar el margen de ruido en alto (NM(H) = 0,7V):

$$V_{CE(off)} > V_{IHmin} + NM(H)$$

$$V_{CE(off)} > 2{,}7V$$



$$V_{C(min)} - V_{R2} - V_{CE(off)} = 0$$

$$V_{CE(off)} = V_{C(min)} - V_{R2}$$

$$V_{CE(off)} = V_C - R2 \cdot (I_{CEOmax} + I_{IHmax})$$

Aplicando la condición anterior se obtiene el valor máximo para R2:

$$V_{C(min)} - R2 \cdot (I_{CEOmax} + I_{IHmax}) > 2{,}7V$$

$$R2 < \frac{V_{C(min)} - 2{,}7}{I_{CEOmax} + I_{IHmax}} = \frac{4{,}75 - 2{,}7}{(50 \times 10^{-9}) + (20 \times 10^{-6})} = 102{,}2k\Omega$$

Por lo tanto, de este apartado y del anterior se obtiene:

$$3{,}9k\Omega < R2 < 102{,}2k\Omega$$

El valor de R2 elegido determinara en parte la velocidad máxima del circuito. En las conmutaciones de bajo a alto del fototransistor, la capacidad C_I de entrada del inversor 74LS04 se carga a través de R2, de forma que a mayor R2 mayor es el tiempo de conmutación. Además, hay que tener en cuenta la velocidad de conmutación del optoacoplador que tiene unos tiempos de subida y bajada del orden de 2µs. Para este caso, como no existe ninguna restricción temporal, se eligió R2 = 10k**Ω**.

d) Protección LED

En este último apartado se calculará el valor de la resistencia R3 para la protección del LED1 o indicador luminoso de estado. Estos cálculos se llevaron a cabo teniendo en cuenta que, para el led su tensión umbral es de 1.8V y la máxima corriente que puede circular por él es de 20mA; aun así, con la intención de dejar un margen de seguridad se asumirá una corriente de 17mA.

Aplicando estos conceptos a la ley de Ohm, se tiene que:

$$V = I \cdot R$$

$$V_{in(max)} - V_{led} = I_{led} \cdot R3$$

$$R3 = \frac{V_{in(max)} - V_{led}}{I_{led}} = \frac{5{,}25 - 1{,}8}{17 \times 10^{-3}} = 203\Omega$$

Para asegurar que el led brille con mayor intensidad sin el riesgo que se funda, se tomó R3 = 330**Ω**.

Los esquemáticos y diagramas electrónicos completos del módulo de entradas digitales se encuentra en la sección de anexos.

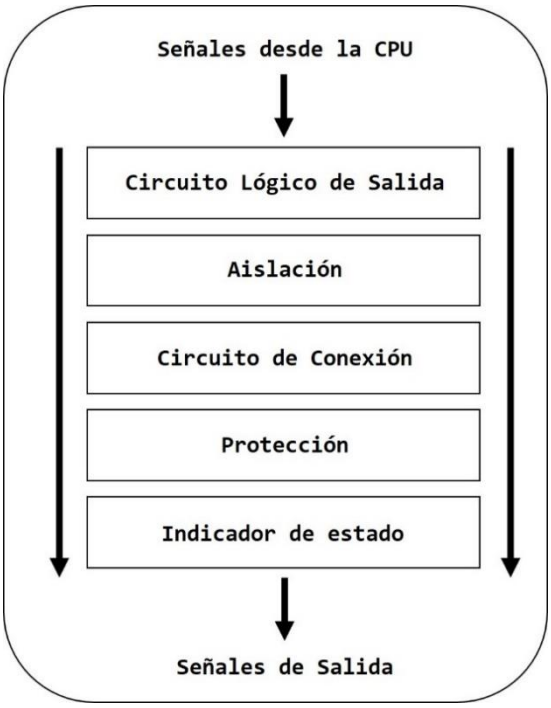
2.1.3 Diseño módulo de expansión salidas digitales

El módulo de salidas digitales tendrá como función principal permitir al PLC actuar sobre los diferentes dispositivos o elementos de control que responden únicamente a señales discretas u ordenes de tipo todo o nada (ON/OFF). Este módulo entregara como salida señales de tensión de 0 ó 24 VDC según el estándar comúnmente utilizado y contara con 8 salidas de tipo Relé.

El diseño de la estructura general de cada una de las salidas digitales se ilustra en la figura 12, la cual se encuentra compuesta por los siguientes bloques:

- ***Circuito lógico de salida:*** Actúa como receptor de la información enviada por la CPU.
- ***Aislación:*** Cumple una función análoga a la de la aislación de una tarjeta de entradas.
- ***Indicador de estado:*** Indica si la salida está activa o desactiva, se utiliza uno por canal.
- ***Circuito de conexión:*** Es el elemento de salida a campo, que maneja la carga conectada por el usuario.
- ***Protección:*** Consiste en un elemento de protección contra sobrecarga de corriente eléctrica

Figura 12. *Diagrama de bloques de una salida discreta*



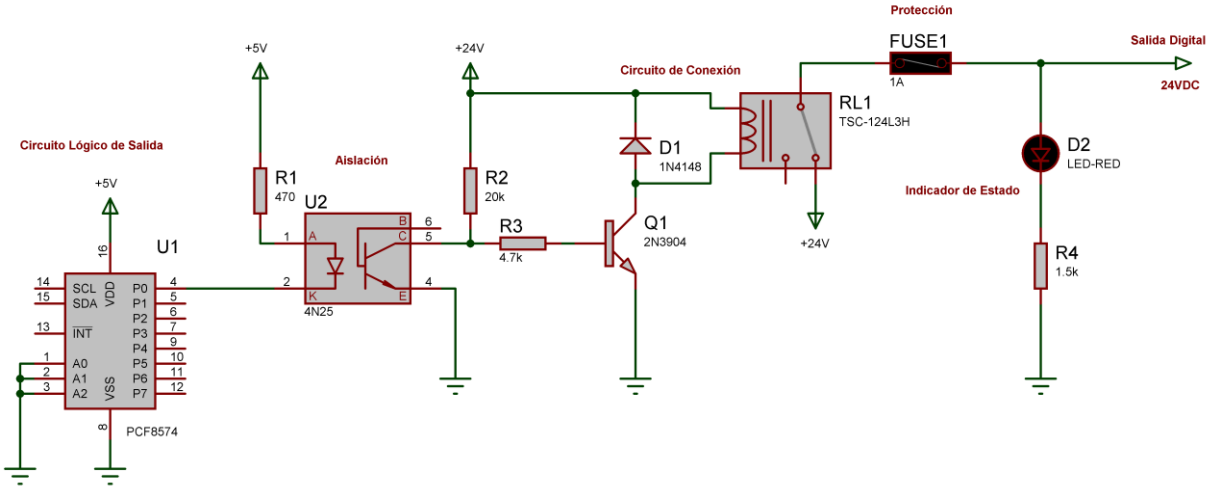
Fuente: Elaboración Propia

A continuación, se explicará el diseño del circuito de una salida digital del prototipo; en la figura 13 se observa el diagrama electrónico correspondiente.

Para poder manejar las 8 salidas digitales se utilizó un segundo PCF8574, pero en este caso se configuraron sus pines como puerto de salida y se seleccionó la dirección de esclavo 0x20 para la comunicación con la CPU mediante el bus I2C; las conexiones de los pines de direccionamiento A0, A1 y A2 se llevaron a cabo de acuerdo a la tabla 5. Según el diagrama de bloques de la figura 12 este dispositivo actuaría como el ***circuito lógico de salida***.

Con la intención de trabajar de acuerdo al nivel de tensión comúnmente utilizado para este tipo de módulos se diseñó una interfaz de salida dividida en varias etapas. La primera etapa corresponde a la etapa de ***aislación***; esta etapa en forma similar al módulo de entradas digitales se encuentra compuesta por un optoacoplador 4N25 y tendrá como función acoplar la señal de salida del PCF8574 (5V) al nivel manejado por el ***circuito de conexión*** (24V), que corresponde a la segunda etapa conformada por un transistor 2N3904, un diodo 1N4148 y un relé.

Figura 13. *Diagrama electrónico de una salida digital*



Fuente: Elaboración Propia

Como se observa en la figura 13 la resistencia R1 se encargará de limitar la corriente que pasará por el diodo de luz del optoacoplador y la resistencia R2 hará trabajar al fototransistor como un interruptor; asegurando que cuando este se encuentre cerrado polarice el transistor 2N3904 y active el relé, reflejando la tensión de salida (24V) procedente de la fuente de alimentación a través de sus contactos. Por el contrario, si este permanece abierto el transistor no conducirá y el relé no se activará, siendo la señal de salida cero.

La resistencia R3 por otro lado, garantizara una polarización del transistor en la región de saturación, permitiendo que este entregue la suficiente corriente para activar el relé; pero debido a que durante la desactivación de relé una tensión muy elevada de polaridad opuesta se induce en la bobina durante un pequeño lapso de tiempo, se conectó como protección en paralelo a la misma un diodo rectificador inversamente polarizado (1N4148), de tal modo que este absorba los picos de tensión de polaridad opuesta y se eviten daños en el transistor de control.

Finalmente, para la etapa general de ***protección*** de toda la interfaz de salida se empleó un fusible de 1A conectado en serie con los contactos de salida, y como ***indicador de estado*** un LED con su respectiva resistencia (R4).

Los valores respectivos de R1, R2, R3 y R4 para que el circuito de salida funcione correctamente se calcularon utilizando los siguientes datos y aplicando las formulas o condiciones usadas con anterioridad para el diseño de la interfaz de entradas digitales, de la siguiente forma:

$V_{CC1} = 5V \pm 5\%$

$V_{CC2} = 24V$

$PCF8574$: $I_{IHmax} = 400\mu A$ $I_{ILmax} = -0{,}4mA$ $I_{OLmax} = 10mA$

$V_{IHmin} = 3{,}5V$ $V_{ILmax} = 1{,}5V$ $V_{OLmax} = 0{,}5V$

$I_{OHmax} = 300\mu A$ $V_{OHmin} = 2{,}5V$

$4N25$: $CTR_{min} = 0{,}2$ $V_{Fmin} = 1{,}2V$ $V_{Fmax} = 1{,}5V$

$I_{Fmax} = 50mA$ $I_{CEOmax} = 50nA$ $V_{CE(sat)max} = 0{,}5V$

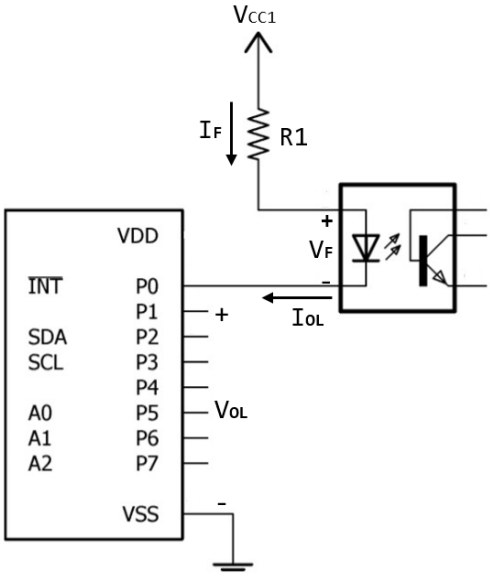
a) Elección de R1

Como se sabe la cantidad de corriente (I_F) que circulara por el led del optoacoplador dependerá del valor de R1, pero para que este emita la suficiente luz mientras conduce I_F tendrá que cumplir las siguientes dos condiciones:

$$I_F < I_{Fmax} = 50mA$$

$$I_F < I_{OLmax} = 10mA$$

Para este caso la primera condición viene impuesta por el led del optoacoplador y la segunda por la capacidad de absorber intensidad en bajo del PCF8574 (I_{OLmax}). Según esto la corriente I_F se calculará de acuerdo al siguiente análisis:



$$I_F = \frac{V_{CC1} - V_F - V_{OL}}{R1}$$

Aplicando la segunda condición impuesta por el PCF8574 a la ecuación anterior se obtiene el límite inferior para R1.

$$I_F < I_{OLmax}$$

$$\frac{V_{CC1(max)} - V_{Fmin} - V_{OLmin}}{R_1} < I_{OLmax}$$

$$R_1 > \frac{V_{CC1max} - V_{Fmin} - V_{OLmin}}{I_{OLmax}}$$

$$R1 > \frac{5{,}25 - 1{,}2 - 0}{10 \times 10^{-3}} = 405\Omega$$

Se eligió un valor comercial para la resistencia R1 = 470Ω.

Según se observa en la figura 13, la señal de salida del expansor se acoplará a la base de un transistor 2N3904, mediante el optoacoplador 4N25. Para asegurar el estado de conmutación del fototransistor para este caso fue necesario considerar tanto los datos técnicos del optoacoplador, como los del transistor 2N3904 especificados a continuación; y así calcular el valor de R2.

$2N3904$: $V_{BE} = 0{,}7V \quad I_B = 1mA \quad h_{FE} = 50$

$I_C = 10mA \quad V_{CE} = 1V$

b) Fototransistor saturado

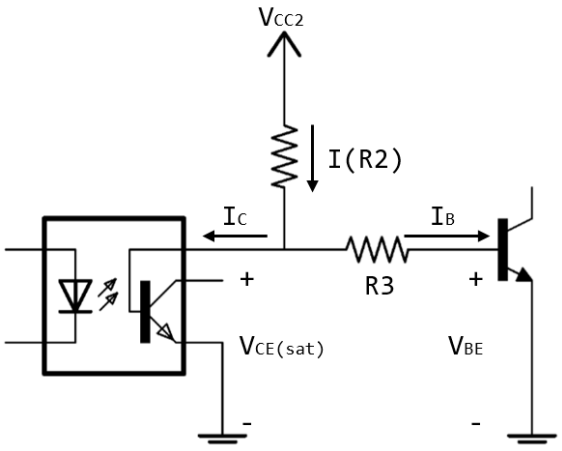
Cuando el fototransistor este saturado la tensión a su salida ($V_{CE(sat)}$) será menor que el voltaje base-emisor (V_{BE}) del transistor 2N3904.

$$\left(V_{CE(sat)max} = 0{,}5V\right) < (V_{BE} = 0{,}7V)$$

Nuevamente para que el fototransistor este saturado se debe cumplir la siguiente condición:

$$I_{C(\max)} < CTR\ .\ I_{F(\min)} = 0{,}2 \cdot I_{F(\min)}$$

En primer lugar, se debe establecer la corriente I_C la cual depende directamente de la resistencia R2.



$$I_C = I(R2) - I_B$$

$$I_{C(max)} = \frac{V_{CC2} - V_{CE(min)}}{R2} - I_B$$

Por otra parte, el valor mínimo de IF se obtiene a través del análisis realizado en el apartado anterior como:

$$I_{F(min)} = \frac{V_{CC1} - V_{Fmax} - V_{OLmax}}{R1} = \frac{5 - 1{,}5 - 0{,}5}{470} = 6{,}4mA$$

Ahora imponiendo la condición de saturación del fototransistor podemos despejar el valor mínimo de R2:

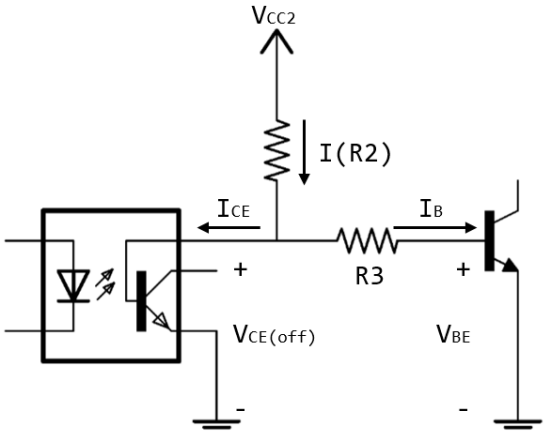
$$\frac{V_{CC2} - V_{CE(min)}}{R2} - I_B < 0{,}2 \cdot I_{F(\min)}$$

$$R2 > \frac{V_{CC2} - V_{CE(min)}}{0{,}2 \cdot I_{F(\min)} + I_B} = \frac{24 - 0}{(0{,}2 \cdot 6{,}4 \times 10^{-3}) + (1 \times 10^{-3})} = 10{,}5k\Omega$$

c) Fototransistor en corte

Por último, cuando el fototransistor este en corte, la tensión VCE(off) será mayor que el voltaje base-emisor del transistor 2N3904.

$$V_{CE(off)} > V_{BE} = 0{,}7V$$



$$V_{CC2} - V_{R2} - V_{CE(off)} = 0$$

$$V_{CE(off)} = V_{CC2} - V_{R2}$$

$$V_{CE(off)} = V_{CC2} - R2 \cdot (I_{CEOmax} + I_B)$$

Aplicando la condición anterior se halló el valor máximo de R2:

$$V_{CC2} - R2 \cdot (I_{CEOmax} + I_B) > 0{,}7V$$

$$R2 < \frac{V_{CC2} - 0{,}7}{I_{CEOmax} + I_B} = \frac{24 - 0{,}7}{(50 \times 10^{-9}) + (1 \times 10^{-3})} = 23{,}3k\Omega$$

De este apartado y del anterior se obtuvo que:

$$10{,}5k\Omega < R2 < 23{,}3k\Omega$$

Como se mencionó anteriormente el valor de R2 determinara la velocidad de conmutación del optoacoplador, pero debido a que para este diseño no se tuvo en cuenta ninguna restricción temporal se eligió R2 = 20kΩ.

d) Polarización del transistor 2N3904

Para obtener una correcta activación del relé es necesario que el transistor se encuentre saturado, es decir, que permita pasar toda la corriente posible como si fuera un simple interruptor cerrado. Esto solo se cumplirá si la corriente de la base es igual a 1mA según se definió con anterioridad de acuerdo a los datos técnicos del transistor 2N3904.

Básicamente para controlar el relé es necesario tener en cuenta las siguientes consideraciones:

- La base del transistor debe superar los 0,6V para que este entre en conducción
- La corriente que pasa entre emisor y colector depende de la corriente que entra por la base multiplicada por la ganancia de corriente del transistor (h_{FE}).

Sabiendo esto se puede obtener la resistencia de base así:

$$R = \frac{(V_{in} - 0{,}6) \cdot h_{FE}}{I_{relé}}$$

Primero se debe asegurar que la corriente que entrará por la base sea igual a 1mA, de acuerdo al modelo de transistor utilizado la ganancia de corriente es de 50 y según los datos técnicos del relé la corriente que consumirá su bobina será 50mA:

$$I_B = \frac{I_{relé}}{h_{FE}} = \frac{50 \times 10^{-3}}{50} = 1mA$$

Como se puede ver la corriente de la base es igual al valor establecido. Ahora mediante la primera ecuación se hallará el valor de R3, pero antes se debe tener en cuenta que el voltaje de entrada para este caso es $V_{CE(off)}$.

$$V_{CE(off)} = V_{CC2} - R2 \cdot (I_{CEOmax} + I_B)$$

$$V_{CE(off)} = 24 - (20 \times 10^3 \cdot ((50 \times 10^{-9}) + (1 \times 10^{-3}))) = 4.99V$$

Reemplazando finalmente se obtiene que:

$$R3 = \frac{(V_{CE(off)} - 0{,}6) \cdot h_{FE}}{I_{relé}} = \frac{(5 - 0{,}6) \cdot 50}{50 \times 10^{-3}} = 4.4k\Omega$$

Se selecciono un valor comercial para la resistencia R3 = 4.7k**Ω**.

e) Protección LED

Finalmente se halló el valor de la resistencia R4 para la protección del LED o indicador luminoso de estado.

$$V = I \cdot R$$

$$V_{out} - V_{led} = I_{led} \cdot R3$$

$$R4 = \frac{V_{out} - V_{led}}{I_{led}} = \frac{24 - 1{,}8}{17 \times 10^{-3}} = 1{,}3k\Omega$$

Se selecciono un valor comercial para la resistencia R4 = 1.5kΩ.

Los esquemáticos y diagramas electrónicos completos del módulo de salidas digitales se encuentra en la sección de anexos.

2.1.4 Diseño módulo de expansión entradas y salidas analógicas.

Debido a que originalmente el PLC fue diseñado para el control de estados lógicos siendo básicamente un equipo de tecnología digital, la única forma de que pueda trabajar con valores análogos, es que estos se representen internamente por medio de números binarios; para integrar esta funcionalidad al OpenPi Controller se diseñó este último módulo, el cual contiene en una sola placa una tarjeta de entradas analógicas y una tarjeta de salidas analógicas. La primera permite al PLC leer señales de tensión provenientes de un sensor y convertirlas en un valor binario, mientras que la segunda emite una señal analógica de tensión de acuerdo a un valor binario.

2.1.4.1Entradas Analógicas.

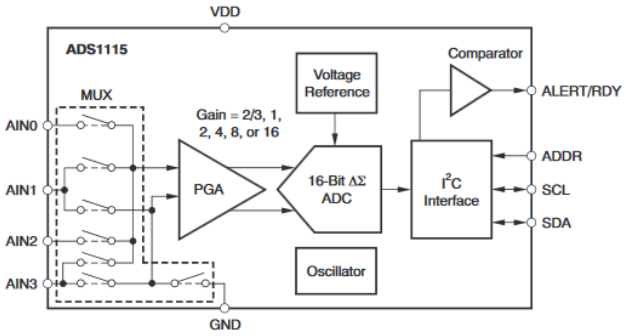
Este módulo se basó en el conversor analógico-digital ADS1115, que se escogió por su facilidad de configuración e integración como capa de hardware, su ganancia programable, y está incluido en la lista de librerías WiringPi, dejando a disposición una serie de funciones para las realizar operaciones de lectura (Ver figura 14).

Figura 14. *Conversor analógico/digital ADS1115*

Fuente: https://www.ittgroup.ee/en/converters/704-analog-digital-converter-16-bit-ads1115.html

Como se aprecia en la figura 15 el ADS115 posee cuatro canales que pueden ser usados como 2 entradas diferenciales o 4 independientes, multiplexadas en ambos casos. Posteriormente se llevan a un PGA, que es un amplificador de ganancia programable, cuya ganancia se controla por medio de una señal externa, que para este caso corresponde al I2C[13].

Figura 15. *Diagrama de bloques funcional del ADS1115*



Fuente: https://cdn-shop.adafruit.com/datasheets/ads1115.pdf

Finalmente, estas señales son procesadas por el ADC de 16 bits, y la información es enviada como una serie de datos, hacia el módulo central del OpenPi Controller, por medio de la interfaz I2C. La dirección que se seleccionó para el conversor fue la 0x49 según la tabla 7, la cual se establece conectando el pin ADDR a VCC.

Tabla 7. *Direccionamiento I2C del ADS1115*

Pin ADDR	Dirección de Esclavo
GND	0x48 (1001000)
VDD	0x49 (1001001)
SDA	0x4A (1001010)
SCL	0x4B (1001011)

Fuente: https://cdn-shop.adafruit.com/datasheets/ads1115.pdf

Esta parte del módulo contara con cuatro entradas analógicas al utilizar los cuatro canales del ADS1115 de manera independiente. Ahora bien, debido a que este conversor soporta un rango de entrada de ±6.144V (Ver tabla 8) y en el estándar se emplean señales de salida de 0-10V, fue necesario diseñar una circuitería externa para adaptar esta señal a los valores que soporta el circuito integrado.

[13] Disponible en la web: https://cdn-shop.adafruit.com/datasheets/ads1115.pdf

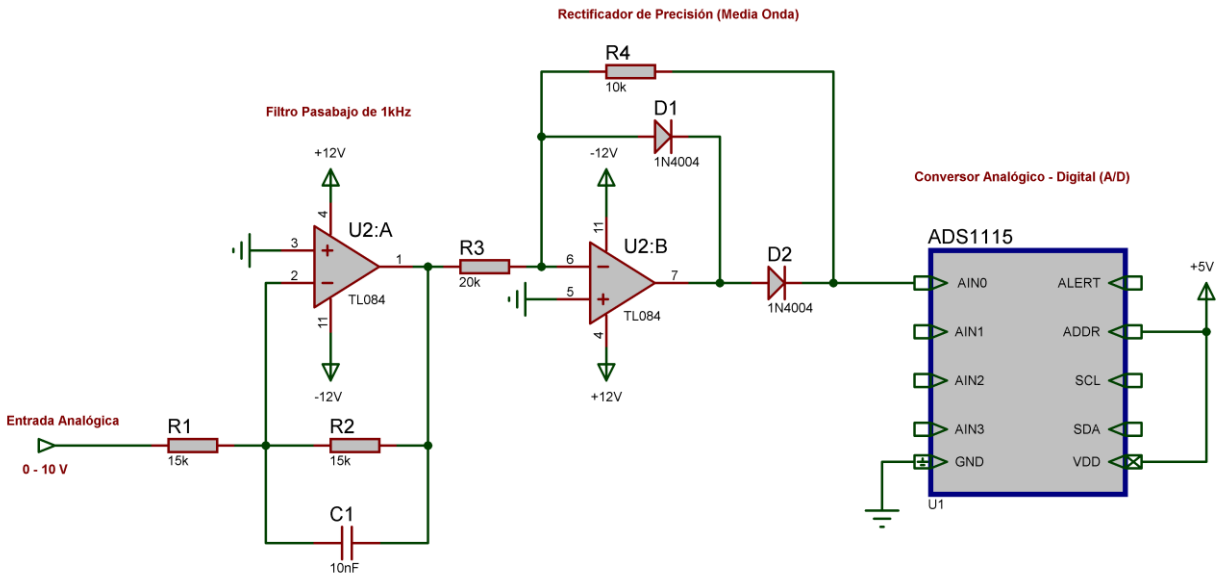
Tabla 8. *Características eléctricas del ADS1115*

Características Eléctricas	
Voltaje de Alimentación	2.0 V a 5.5 V
Consumo de corriente	150µA (típica)
Tasa de datos programable	8 – 860 SPS
Voltaje de Entrada analógico	±6.144V
Resolución	16 bits
Rango de temperatura de operación	-40 a +125°C

Fuente: https://cdn-shop.adafruit.com/datasheets/ads1115.pdf

Se implemento un filtro activo y una configuración de amplificadores operacionales como rectificador de precisión, como se muestra en la figura 16.

Figura 16. *Diagrama electrónico de una entrada analógica*



Fuente: Elaboración Propia

Las señales analógicas que se miden suelen tener un considerable nivel de ruido, por lo que antes de que el ADC realice su lectura, son procesadas por medio de un filtro activo pasa-bajo de ganancia unitaria, es decir, un amplificador inversor cuya frecuencia de corte se estableció a aproximadamente 1kHz. Es también importante mencionar que la señal de entrada estará limitada a los voltajes de alimentación del amplificador operacional, por lo que actúan a su vez como un recortador de señal, en caso de que sobrepase estos valores.

Los componentes se calcularon despejando los valores de las fórmulas de frecuencia de corte y ganancia, respectivamente, como se muestra a continuación.

$$F_C = \frac{1}{2\pi R2C1}$$

$$R2 = \frac{1}{2\pi \cdot F_C \cdot C1}$$

$$R2 = \frac{1}{2\pi \cdot (1 \times 10^3) \cdot (100 \times 10^{-9})} \approx 15k\Omega$$

$$A_v = -\frac{R2}{R1}$$

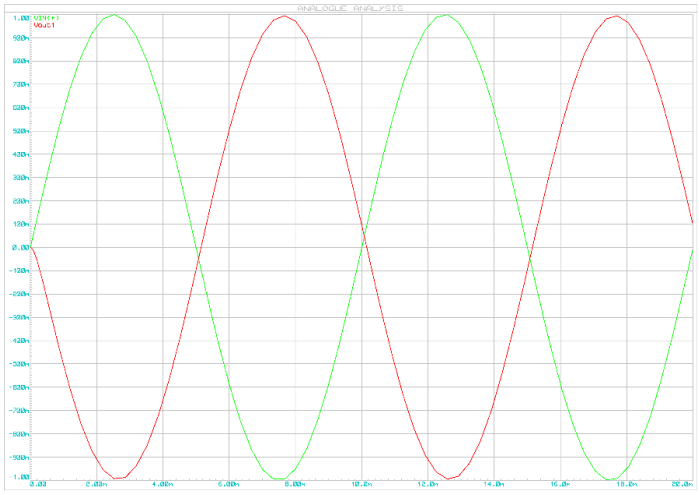
$$-(-1)R1 = R2$$

$$R1 = R2$$

$$R1 = 15k\Omega$$

Debido a que es más fácil conseguir distintos valores de resistores en el mercado, se asumió el valor del capacitor a 10nF.

Figura 17. *Señal de Salida Filtro pasa-bajo*



Fuente: Elaboración Propia

Después de obtener la señal filtrada y desfasada 180° (señal roja), como muestra la simulación en la figura 17, lo siguiente es el proceso de rectificación. Para ello se empleó un rectificador de precisión puesto que la tensión de umbral es muy baja, a diferencia de los 0.7V que requiere un diodo de silicio para realizar la misma función.

Ganancia de un rectificador de precisión (inversor:) $V_{out} = V_{in} \times \left(\frac{-R2}{R1}\right)$

Cálculo valores de resistores.

$$A_v = -\frac{R4}{R3}$$

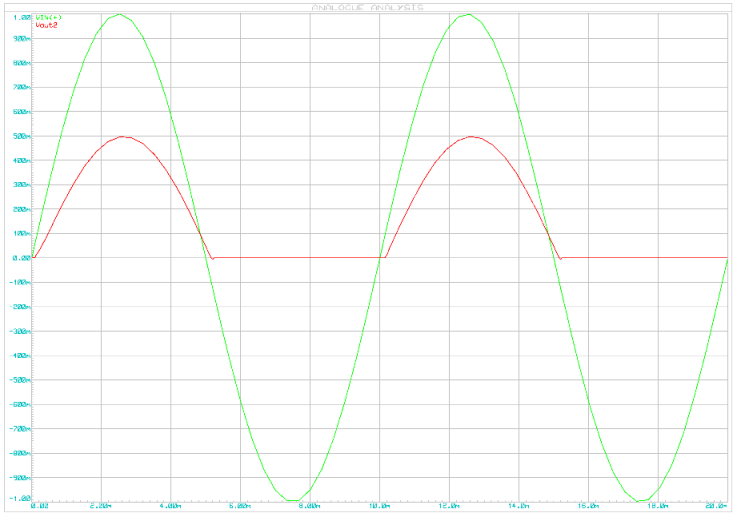
$$-\frac{1}{2} = -\frac{R4}{R3}$$

$$\frac{R1}{2} = R4$$

$$Si\ R3 = 20k\Omega\ \Rightarrow R4 = 10k\Omega$$

Como el requisito de la salida ser una ganancia de -½ puesto que al ADC solamente se le suministrará un rango de valores de 0-5V, y como esta configuración invierte nuevamente la señal, se escoge uno de los valores para los resistores, y el otro se calcula con la fórmula anterior.

Figura 18. *Señal de entrada analógica filtrada y rectificada*



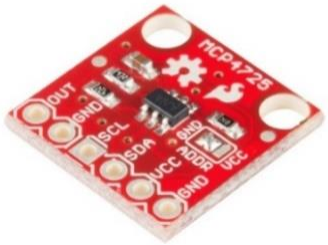
Fuente: Elaboración Propia

La simulación de la figura 18 muestra la señal original de 10V de amplitud (verde), recortada y ahora con una amplitud de 5V (roja), que es el objetivo del diseño inicial.

2.1.4.2 Salida Analógica

Este módulo se basó en el convertidor digital-analógico MCP4725, que posee una resolución de 12 Bits, y al igual que los otros circuitos integrados, se controla mediante I2C (Ver figura 19).

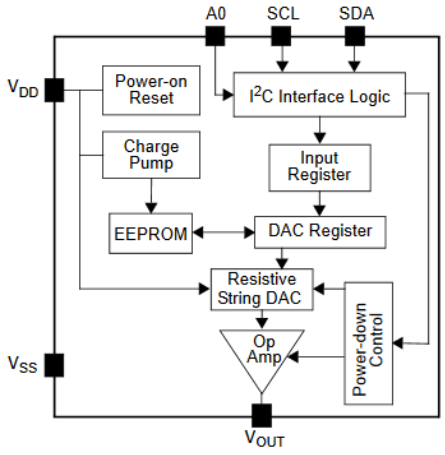
Figura 19. *Convertidor digital-analógico MCP4725*



Fuente: https://learn.sparkfun.com/tutorials/mcp4725-digital-to-analog-converter-hookup-guide

Como se muestra en la figura 20 el MCP4725 posee una memoria volátil EEPROM donde se pueden programar los datos de entrada y configuración del DAC mediante la interfaz I2C, el dispositivo también incluye un circuito Power-On-Reset (POR) para garantizar un encendido seguro y su amplificador de salida de le permite lograr un balance en la tensión analógica de salida (rail to rail), proporcional con el conector de energía[14]. Las demás características del dispositivo se resumen en la tabla 9.

Figura 20. *Diagrama de bloques funcional del MCP4725*



Fuente: https://cdn.sparkfun.com/datasheets/BreakoutBoards/MCP4725_2009.pdf

[14] Disponible en la web: https://cdn.sparkfun.com/datasheets/BreakoutBoards/MCP4725_2009.pdf

Tabla 9. *Características eléctricas del MCP4725*

Características Eléctricas	
Voltaje de Alimentación	2.7 V a 5.5 V
Consumo de corriente	210µA (típica)
Velocidad de transmisión de Datos	100 kbps - 3.4 Mbps
Voltaje de Salida analógico	0 - VDD
Resolución	12 bits
Rango de temperatura de operación	-40 a +125°C

Fuente: https://cdn.sparkfun.com/datasheets/BreakoutBoards/MCP4725_2009.pdf

Como se mencionó anteriormente este conversor cuenta con una selección externa de bits de dirección a través del pin A0, el cual se puede vincular a VDD o GND según las aplicaciones del usuario, permitiéndole utilizar dos dispositivos al tiempo. La dirección de esclavo que se usó por defecto para la comunicación con el OpenPi Controller fue la 0x60, la cual se estableció conectando el pin A0 a GND según la tabla 10.

Tabla 10. *Direccionamiento I2C del MCP4725*

Pin A0	Dirección de Esclavo
GND	0x60 (1100000)
VDD	0x61 (1100001)

Fuente: https://cdn.sparkfun.com/datasheets/BreakoutBoards/MCP4725_2009.pdf

El MCP4725 puede entregar voltajes que van desde 0 hasta 5V de manera proporcional al número entero que recibe por el canal I2C y que oscila entre 0 y 4095. Debido al estándar que se tiene como requerimiento, es decir, un rango de salida 0-10V, se diseñó una interfaz de potencia elemental que consiste en un amplificador operacional en configuración no inversora, con ganancia de 2 (Ver figura 21).

$$A_v = 1 + \frac{R2}{R1}$$

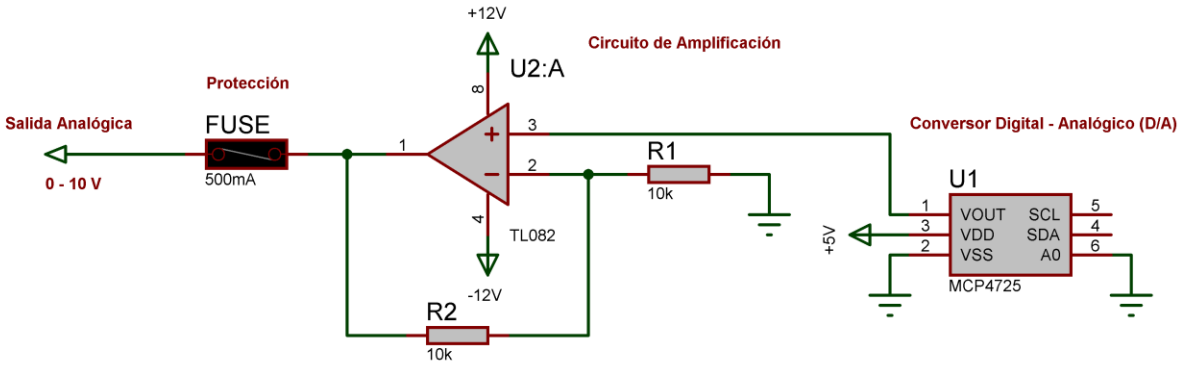
$$2 = 1 + \frac{R2}{10k\Omega}$$

$$(2 - 1) \cdot 10k\Omega = R2$$

$$R2 = 10k\Omega$$

El cálculo de los resistores, se basa en la fórmula de la ganancia para la configuración escogida, como se muestra en la formula anterior. Se asumió un valor comercial de 10kΩ para R1 y se despejo el valor de R2.

Figura 21. *Diagrama electrónico de la interfaz de Salida Analógica de 0 a 10 V*



Fuente: Elaboración Propia

Finalmente, se coloca un fusible de 500mA para proteger de una carga excesiva a la salida de la interfaz.

Los esquemáticos y diagramas electrónicos completos del módulo de entradas y salidas analógicas se encuentra en la sección de anexos.

2.1.5 Fuente de Alimentación

Para la fuente de alimentación del PLC se empleó una fuente poder conmutada AC-DC de 24V/6A (Ver figura 22).

Figura 22. *Fuente de alimentación conmutada AC-DC 24V/6A*

Fuente: http://www.mactronica.com.co/fuente-acdc-24v-6a-107869044xJM

Se utilizo esta placa prefabricada ya que por su tamaño se adaptaba perfectamente a la estructura del prototipo, y sus características cumplen con los requerimientos solicitados para la alimentación de cada una de los diferentes módulos del OpenPi Controller. En la tabla 11 se encuentras las diferentes características de esta fuente.

Tabla 11. *Características eléctricas fuente de alimentación*

Características Eléctricas	
Voltaje de Entrada AC	AC 85-265V (Global common)
Frecuencia	50HZ/60HZ
Voltaje de Salida	DC 24V
Corriente de Salida	4A-6A
Potencia de Salida	100W
Modulación	Modulación de ancho de pulso

Fuente: Elaboración Propia

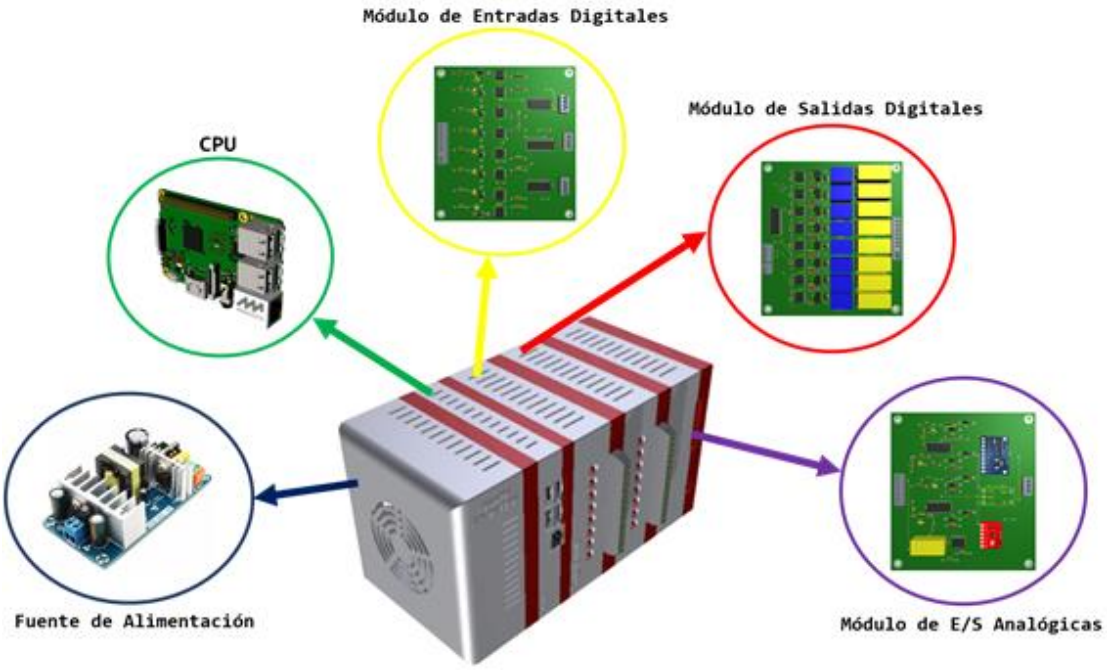
2.1.6 Estructura Física

La estructura externa del OpenPi Controller se basó en una configuración de tipo modular, la cual se caracteriza principalmente porque existe un módulo para cada uno de los elementos que componen al PLC. Este tipo de estructura le ofrece al prototipo la particularidad de ser totalmente extensible, permitiéndole expandirse según lo requiera el proceso que se desee automatizar, con tan solo aumentar el número de entradas y salidas.

Para esta primera versión en desarrollo se cuenta con construcciones separadas para la CPU, fuente de alimentación y sistemas de E/S. Todo esto es posible gracias a la funcionalidad del bastidor o riel, sobre el cual se montarán las placas de circuito impreso contenidas dentro de carcasas correspondientes a cada uno de los módulos explicados anteriormente.

En la figura 23 se esquematiza la estructura general del OpenPi Controller y se observan los diferentes elementos físicos (hardware) que componen el prototipo.

Figura 23. *Estructura externa del OpenPi Controller*

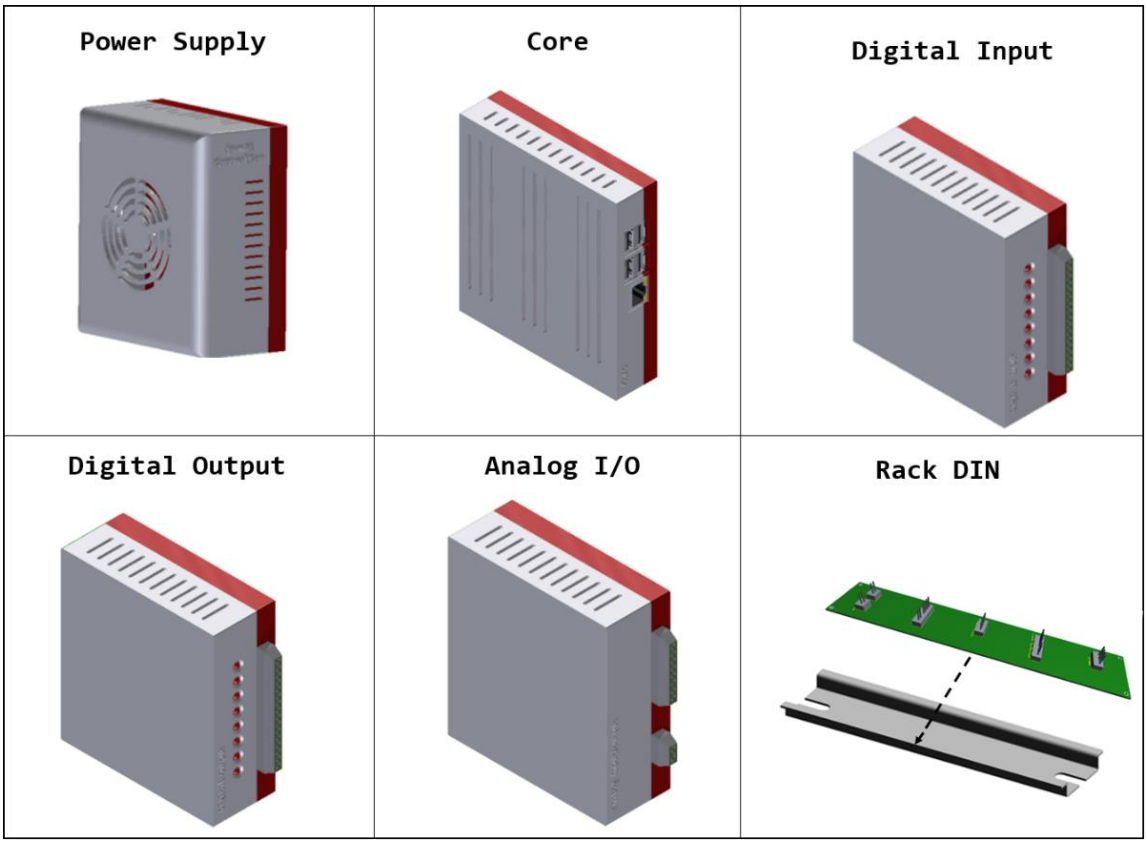


Fuente: Elaboración Propia

Cada uno de las diferentes carcasas de los módulos junto al riel de interconexión del dispositivo se diseñaron en el programa *Autodesk Inventor*, un software de modelado paramétrico de sólidos, para luego ser replicados en una impresora 3D. El material utilizado para la impresión de las diferentes piezas fue plástico ABS (*acrilonitrilo butadieno estireno*) o comúnmente llamado plástico de ingeniería; se empleó debido a sus características de resistencia al impacto y su frecuente uso tanto en aplicaciones industriales como domésticas.

Los archivos CAD y formatos STL de cada una de las carcasas de los módulos del OpenPi Controller se encuentran públicamente disponibles en el repositorio de GitHub que fue creado para este proyecto, junto a los diferentes elementos de software en el siguiente enlace: *https://github.com/Legacier/OpenPiController.* Los archivos se nombraron según como se observa en la figura 24.

Figura 24. *Identificación de los archivos CAD del prototipo OpenPi Controller*



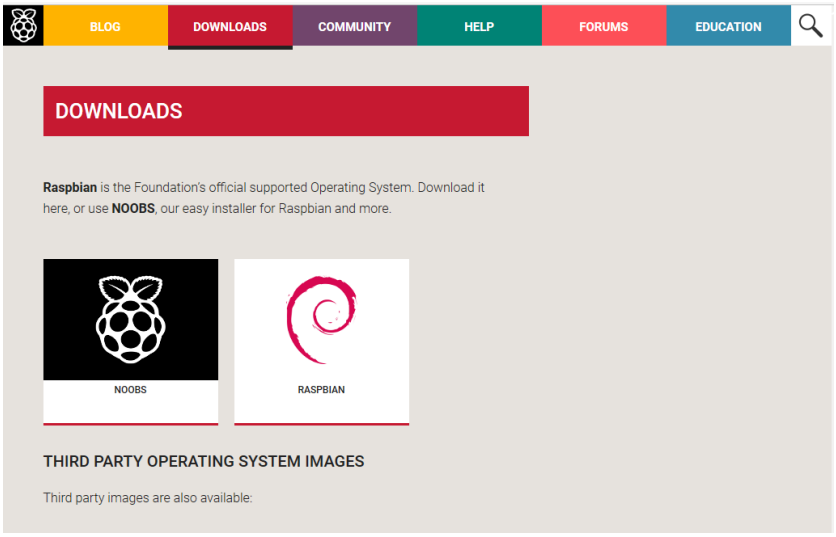
Fuente: Elaboración Propia

2.2 DISEÑO DEL SOFTWARE

2.2.1 Configuración e Instalación de la Raspberry Pi

En primer lugar, se requiere un sistema operativo diseñado especialmente para el dispositivo. Raspbian es una de las múltiples opciones que existe para este fin. Se escogió debido al amplio soporte por parte de la comunidad y la cantidad de programas y paquetes existentes para la plataforma. El sitio web de la Fundación Raspberry Pi (https://www.raspberrypi.org) nos ofrece dos imágenes de instalación en la sección de descargas, cuya diferencia radica en el asistente que incluye cada una de ellas (Ver figura 25).

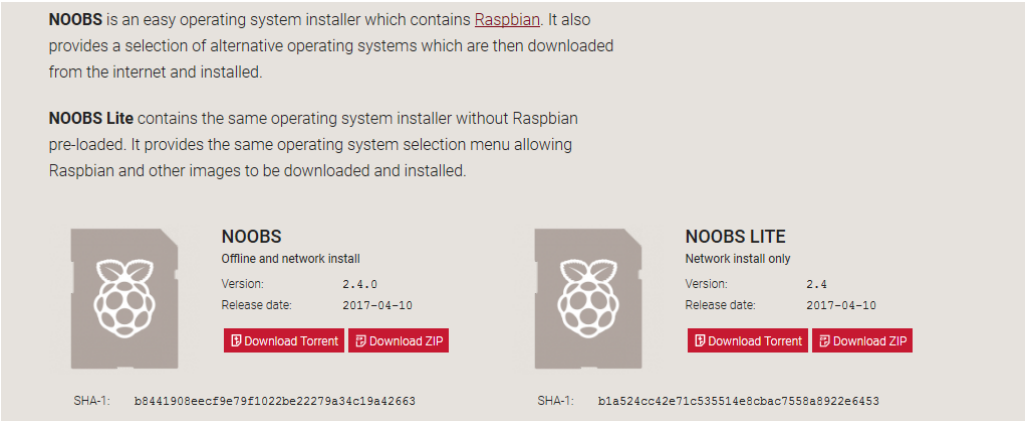
Figura 25. *Página de descargas de RPI*



Fuente: https://www.raspberrypi.org

La versión que posee el asistente de instalación NOOBS, a su vez tiene dos opciones disponibles para su descarga, como se observa en la figura 26. En este caso se usó la versión Offline, puesto que ya incluye los archivos necesarios para el proceso y no necesita descargar paquetes adicionales

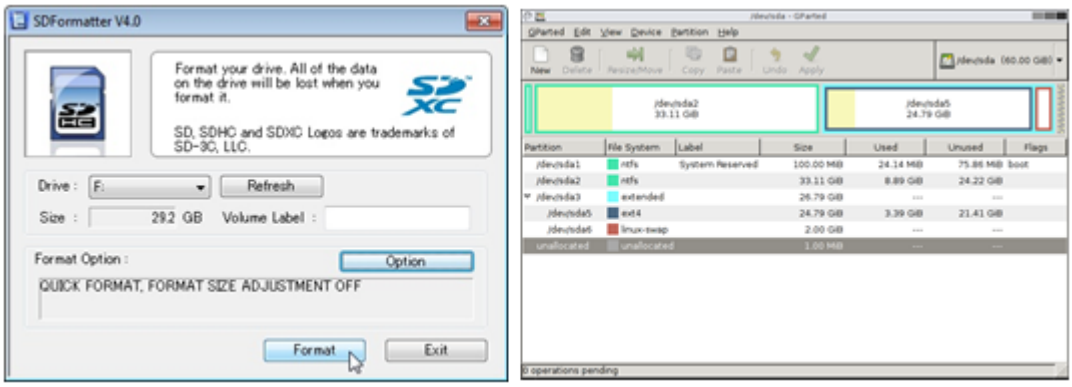
Figura 26. *Versiones de Descarga de NOOBS*



Fuente: https://www.raspberrypi.org

La imagen ISO debe ser copiada a una tarjeta microSD que tenga capacidad mínima de 8GB. Para el desarrollo del proyecto la elección fue de una micro SDHC de 32GB clase 10, con el fin de mejorar ligeramente la velocidad de lectura/escritura, en especial a la hora de realizar actualizaciones mediante el gestor de paquetes. Se requiere previamente formatear la tarjeta de memoria con una utilidad como SD Formatter para Windows, o GParted para distribuciones GNU/Linux (Ver figura 27).

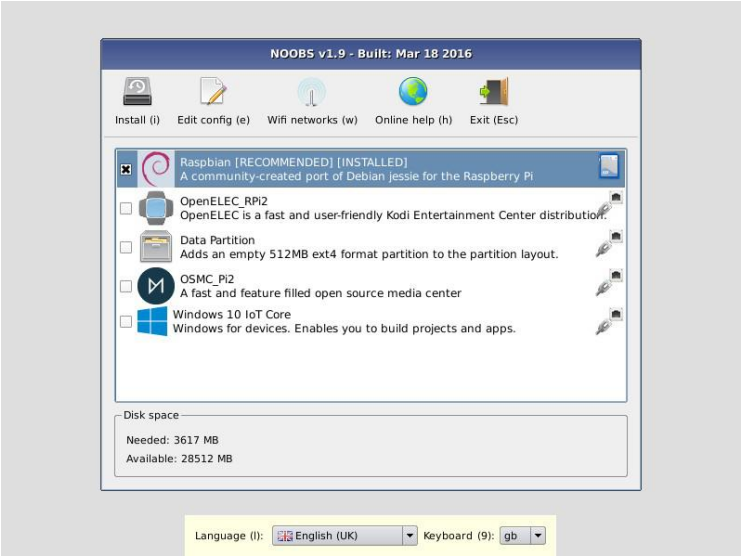
Figura 27. *Herramientas para Formateo de SD*



Fuente: https://www.sdcard.org/downloads/formatter_4/

Posteriormente, al encender la Raspberry Pi, se muestra un menú con varias opciones, donde se eligió Raspbian, como se mencionó de manera previa, en la figura 28 se muestra el proceso de selección.

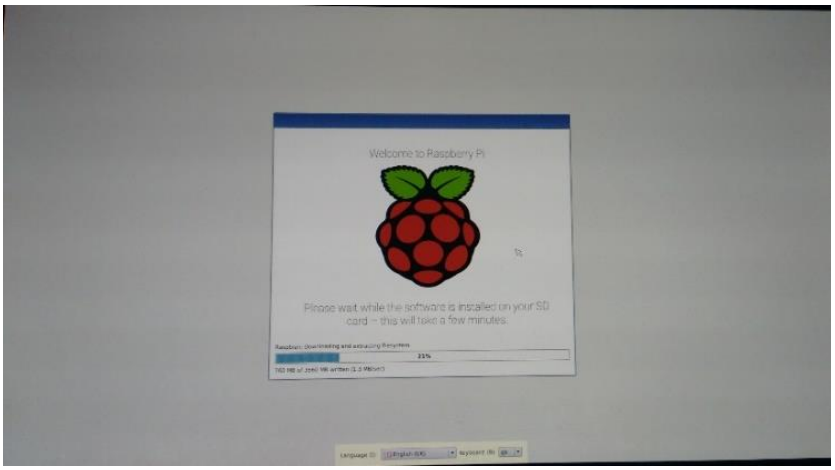
Figura 28. *Selección de Sistema Operativo a instalar*



Fuente: Elaboración Propia

El asistente se encarga de realizar las configuraciones necesarias para tener un sistema y entorno de escritorio listo para su uso al final del proceso (Ver figura 29).

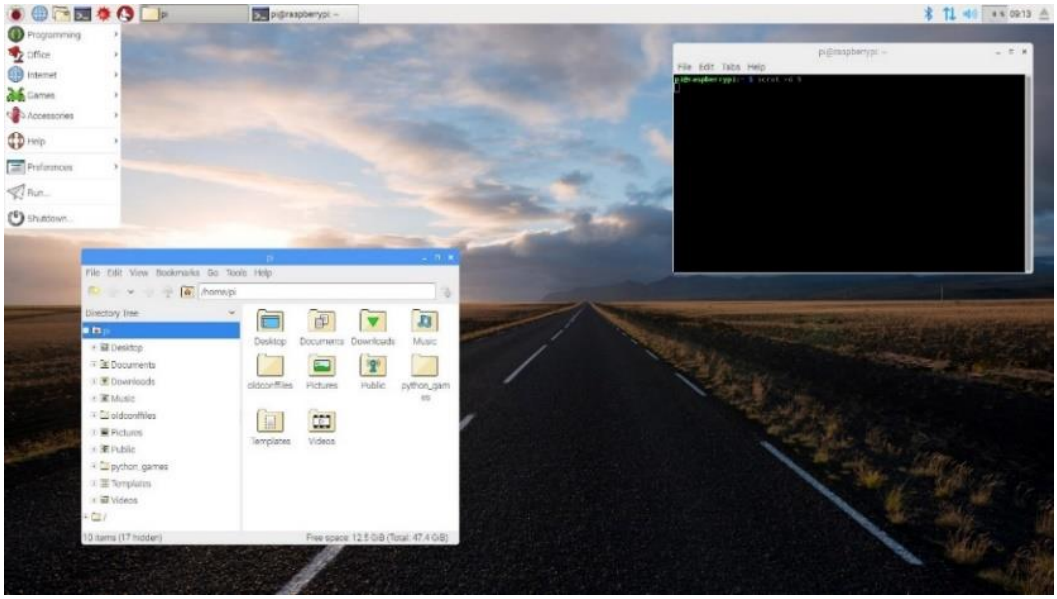
Figura 29. *Progreso de Instalación del Sistema Operativo NOOBS*



Fuente: Elaboración Propia

En la figura 30 se aprecia el entorno de escritorio que contiene una serie de herramientas y aplicaciones para facilitar el uso de distintos lenguajes de programación, y algunas utilidades para habilitar y deshabilitar interfaces de la Raspberry Pi tales como el acceso SSH, I2C, entre otros.

Figura 30. *Entorno de Escritorio*



Fuente: Elaboración Propia

2.2.2 Instalación de OpenPLC

El propósito de usar OpenPLC en el proyecto, fue el de emular un PLC en un hardware que funcione bajo GNU/Linux[15]. Éste PLC puede ejecutar programas basados en el estándar IEC-61131-3, responde a solicitudes Modbus/TCP y posee una gran facilidad de integrar dispositivos físicos tales como expansores, conversores ADC y DAC. Adicionalmente, contiene de una capa de hardware que usa C++ como lenguaje de programación.

El sitio oficial del proyecto muestra una serie de paquetes y dependencias que deben ser instalados antes de que el software pueda funcionar. WiringPi es una librería para facilitar el acceso al GPIO de la RaspberryPi, escrita en C y disponible en otra gran cantidad de lenguajes, que está diseñado para que programar las entradas y salidas, sea muy similar a la manera en cómo se hace en un Arduino[16]. En las imágenes de instalación más recientes, WiringPi se incorporó por defecto; los demás paquetes se obtuvieron mediante el gestor APT, desde los repositorios oficiales, con los siguientes comandos (figura 31).

Figura 31. *Paquetes adicionales*

```
sudo apt-get update
sudo apt-get install build-essential pkg-config bison flex
sudo apt-get install autoconf automake libtool make nodejs git
```

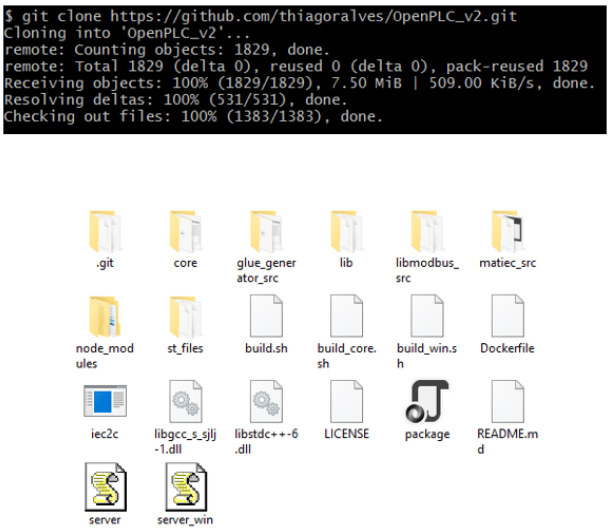
Fuente: Elaboración Propia

A continuación, se clonó el repositorio de OpenPLC que contiene el código fuente del proyecto original y los scripts necesarios para la compilación (Ver figura 32). Se realizó una modificación al script principal y se añadió a la lista de capas de hardware disponible el controlador necesario para que la Raspberry Pi soporte los módulos de expansión previamente diseñados. Estas modificaciones se pueden observar detalladamente en la sección de anexos, correspondientes a los archivos build.sh y openpicontroller.cpp.

[15] T. R. Alves, M. Buratto, F. M. de Souza and T. V. Rodrigues, "OpenPLC: An open source alternative to automation" IEEE Global Humanitarian Technology Conference (GHTC 2014), San Jose, CA, 2014, pp. 585-589.

[16] WiringPi: GPIO Interface library for the Raspberry Pi. Gordon Projects. Disponible en: http://wiringpi.com

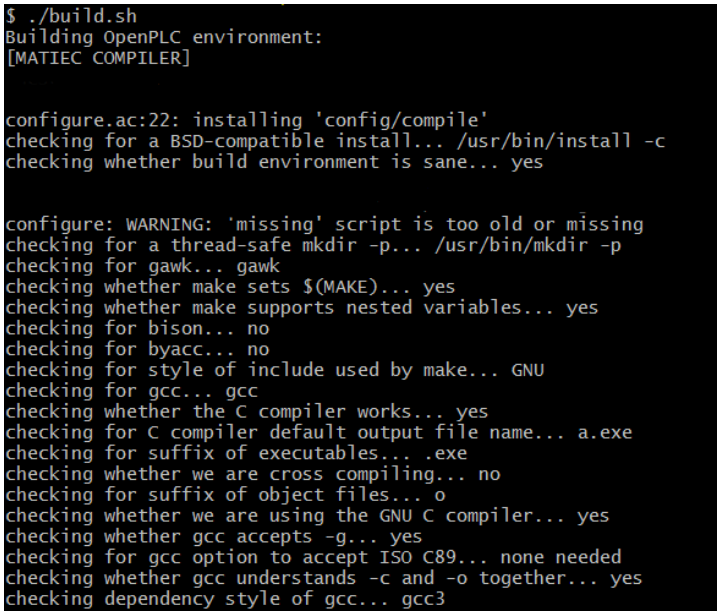
Figura 32. *Clon de Repositorio Oficial*



Fuente: Elaboración Propia

Al ejecutar el archivo build.sh inicia la verificación de librerías que requiere el proceso de compilación. El tiempo de espera es de aproximadamente 5 a 7 minutos para generar el ejecutable requerido para la puesta en marcha de la aplicación. El proceso de compilación que se lleva a cabo en la consola del sistema se muestra en la figura 33.

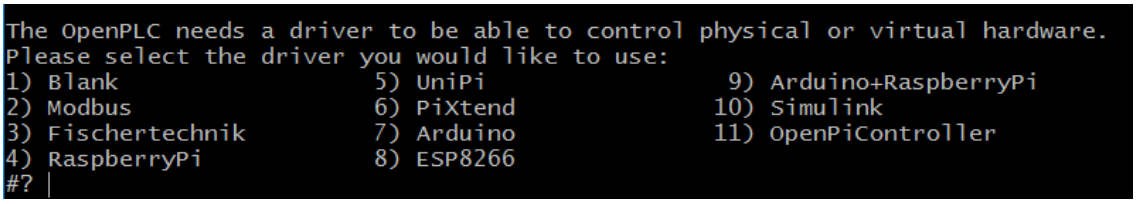
Figura 33. *Proceso de Compilación de OpenPLC*



Fuente: Elaboración Propia

Una vez completo, se muestra un menú como se detalla en la figura 34 con diferentes opciones que se refieren a la capa de hardware que se desea emplear. Para este caso, se escogió la opción de OpenPiController.

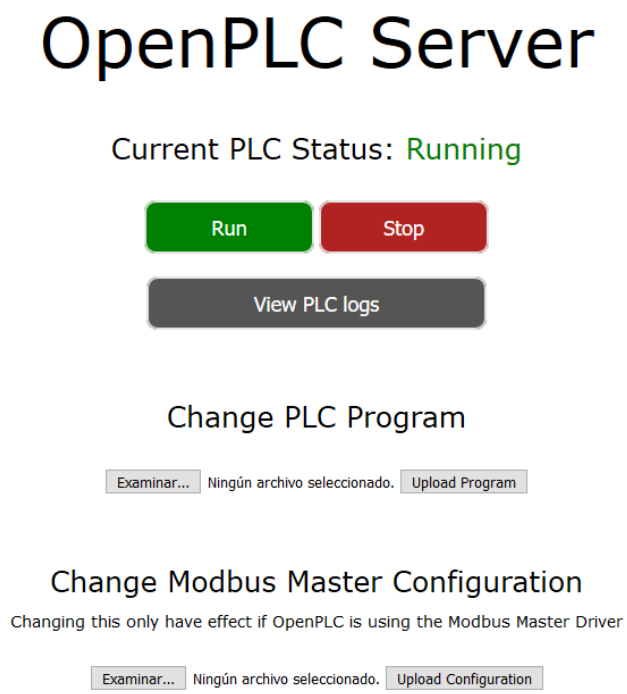
Figura 34. *Selección de Capa de Hardware*



Fuente: Elaboración Propia

Por último, se puede comprobar si la instalación se completó de manera correcta, al acceder a la interfaz web que se genera, mediante cualquier navegador, indicando la dirección IP donde está localizada la Raspberry Pi, y el puerto 8080 que es el usado de manera predeterminada por la aplicación. En la figura 35 se puede ver la interfaz de la aplicación web del servidor de OpenPLC.

Figura 35. *Interfaz Web de OpenPLC*

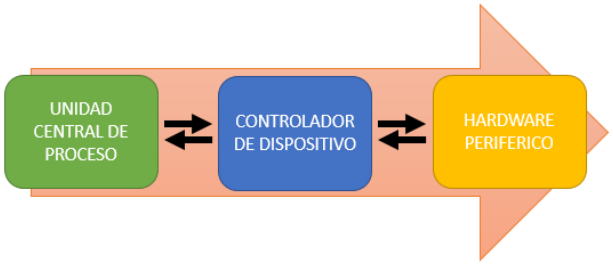


Fuente: Elaboración Propia

2.2.3 Diseño de la Capa de Hardware de OpenPi Controller

La capa de hardware será el elemento del sistema operativo que funcionara como una interfaz entre el software (OpenPLC) y el hardware (OpenPi Controller) del sistema, proporcionando una plataforma de hardware consistente sobre la cual correrán los programas o rutinas del prototipo de PLC (Ver figura 36).

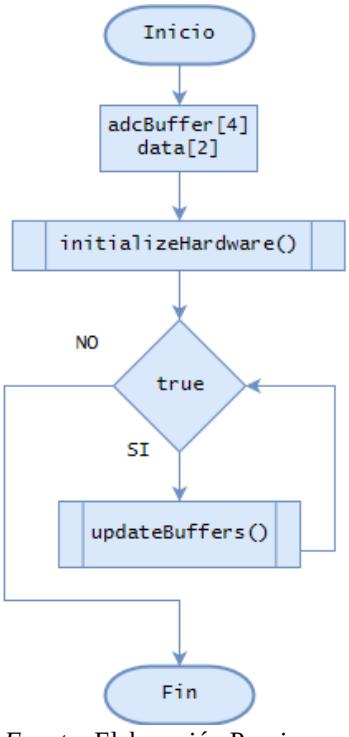
Figura 36. *Interacción de los componentes de software y hardware del PLC*



Fuente: Elaboración Propia

El orden de operaciones que se llevan a cabo al inicializar OpenPLC se ilustran en el diagrama de flujo de la figura 37, donde se tienen 2 búfer en los que se almacenan los datos de las lecturas del ADC y las órdenes que serán enviadas al DAC.

Figura 37. *Diagrama de Rutina Principal de OpenPLC*



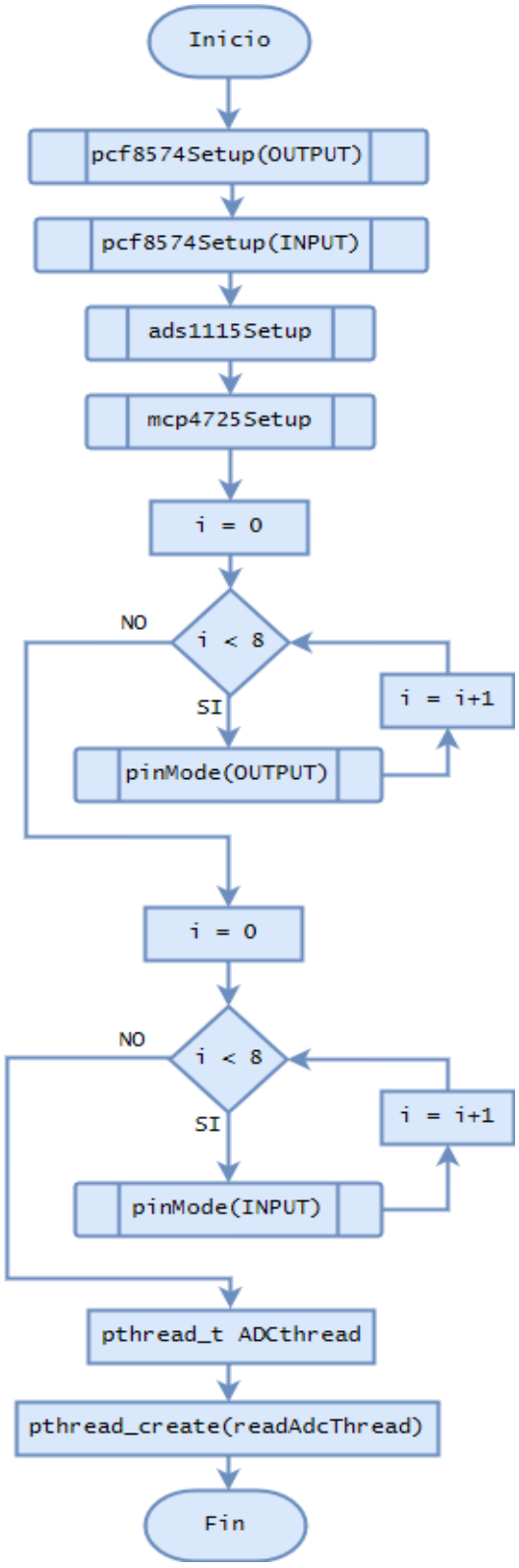
Fuente: Elaboración Propia

Posteriormente, de manera análoga a las funciones ***setup*** y ***loop*** de un Arduino, que están presentes en cada programa elaborado para esa plataforma, OpenPLC posee en las capas de hardware disponibles, una función ***initializeHardware*** y un ciclo indeterminado donde se llamará otra función conocida como ***updateBuffers***.

Cada capa de hardware es en realidad un controlador escrito en lenguaje C++, que le indica al programa principal la forma adecuada de leer y escribir los datos y estados almacenados en los buffers, a los dispositivos físicos que estén conectados. Para el caso del OpenPi Controller, se diseñó un controlador propio para los módulos, que se encuentra en la ruta *core/hardware_layers/* y fue nombrado **openpicontroller.cpp**. El código fuente del controlador se encuentra en los anexos, y disponible para su descarga en el repositorio de GitHub creado para este proyecto.

InitializeHardware es la función encargada de realizar las configuraciones iniciales necesarias. En primer lugar, la asignación de direcciones a cada dispositivo que está conectado por medio de la interfaz I2C, que corresponde a dos expansores GPIO PCF8574, un conversor análogico-digital ADS1115 y un conversor digital-análogico MCP4725, mediante las funciones de setup que se muestran en los bloques de la figura 38. A continuación, asigna la función respectiva de entrada o salida digital a cada pin de los expansores, por medio de la función ***pinMode***. Finalmente se instancia un objeto tipo *pthread* y se inicializa un hilo adicional que está encargado de tomar lecturas de los 4 canales del ADS1115 y almacenarlas en el búfer.

Figura 38. *Diagrama del proceso de Inicialización del Hardware de OpenPi Controller*



Fuente: Elaboración Propia

Por otra parte, ***updateBuffers*** es otra función esencial que es llamada de manera periódica por la rutina principal de OpenPLC. La tarea que se ejecuta, es primero bloqueada por un *mutex_lock*, para garantizar la integridad de los datos en las lecturas. Existen 4 matrices principales que son los buffers donde se almacenan los estados del PLC:

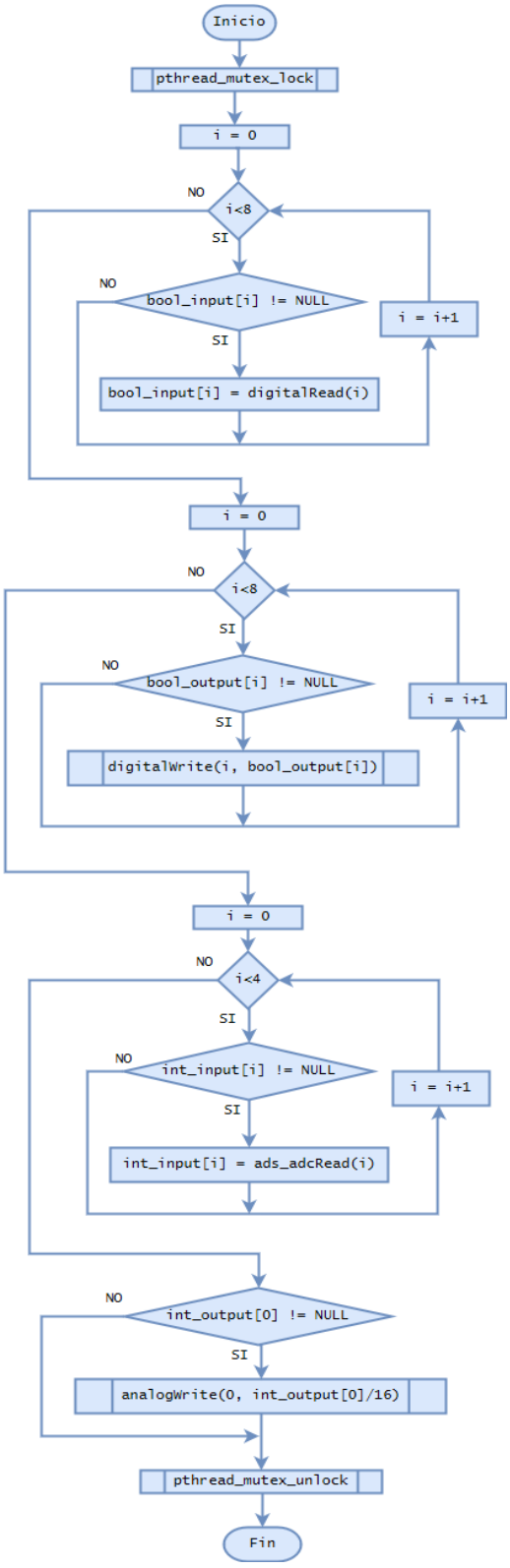
- bool_input para las entradas digitales.
- bool_output para las salidas digitales.
- int_input para las entradas analógicas.
- int_output para las salidas analógicas.

Durante la ejecución de la función, se llevan a cabo dos operaciones de lectura, que corresponden a las entradas digitales y analógicas, por medio de los métodos ***digitalRead*** y ***analogRead***. Se comprueba de antemano que existan datos, razón por la cual se comparan con la constante NULL, y se guardan los estados o el valor entero presente en cada pin, en su búfer correspondiente, es decir, actualiza el valor de cada posición de la matriz.

Para las operaciones de escritura, por el contrario, toma el valor que le es suministrado por el búfer y lo refleja en los pines correspondientes a los circuitos integrados de salidas digital y analógica, llamando a los métodos ***digitalWrite*** y ***analogWrite***. Es importante mencionar, que el búfer de la salida analógica usa el tipo de dato WORD, es decir, un entero de 16 bits; por tal motivo debe ser escalado a un valor de 12 bits, que es la resolución permitida por el DAC, siendo ésta la razón por la que este valor debe dividirse por 16.

Finalmente, se hace un llamado a *mutex_unlock* para liberar el acceso a las posiciones de memoria usadas. Todo el proceso llevado a cabo durante esta función se representa en el diagrama de flujo de la figura 39.

Figura 39. *Diagrama de actualización de Buffers de OpenPi Controller*



Fuente: Elaboración Propia

2.2.4 Integración del OpenPi Controller en la Nube

Con el fin de tener una manera de supervisar y controlar remotamente el PLC, se decidió implementar una solución basada en el servicio Firebase de Google, una plataforma que permite la creación y uso de bases de datos en tiempo real, provee de almacenamiento en la nube, y una gran facilidad de integración de estos servicios a aplicaciones web y dispositivos móviles.

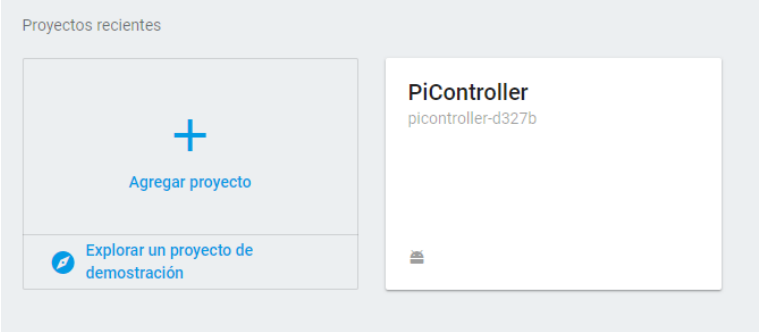
Figura 40. *Sitio oficial de Firebase*

Fuente: Elaboración Propia

Para obtener acceso a estas funcionalidades, es necesario acceder al sitio principal de Firebase (https://firebase.google.com), el cual se observa en la figura 40 y dirigirse a la consola en la parte superior derecha (GO TO CONSOLE).

Allí se muestra una opción que permite crear un nuevo proyecto, además de una lista de otros proyectos que ya estén activos en la cuenta. En la figura 41 se puede apreciar la estructura de la consola de Firebase.

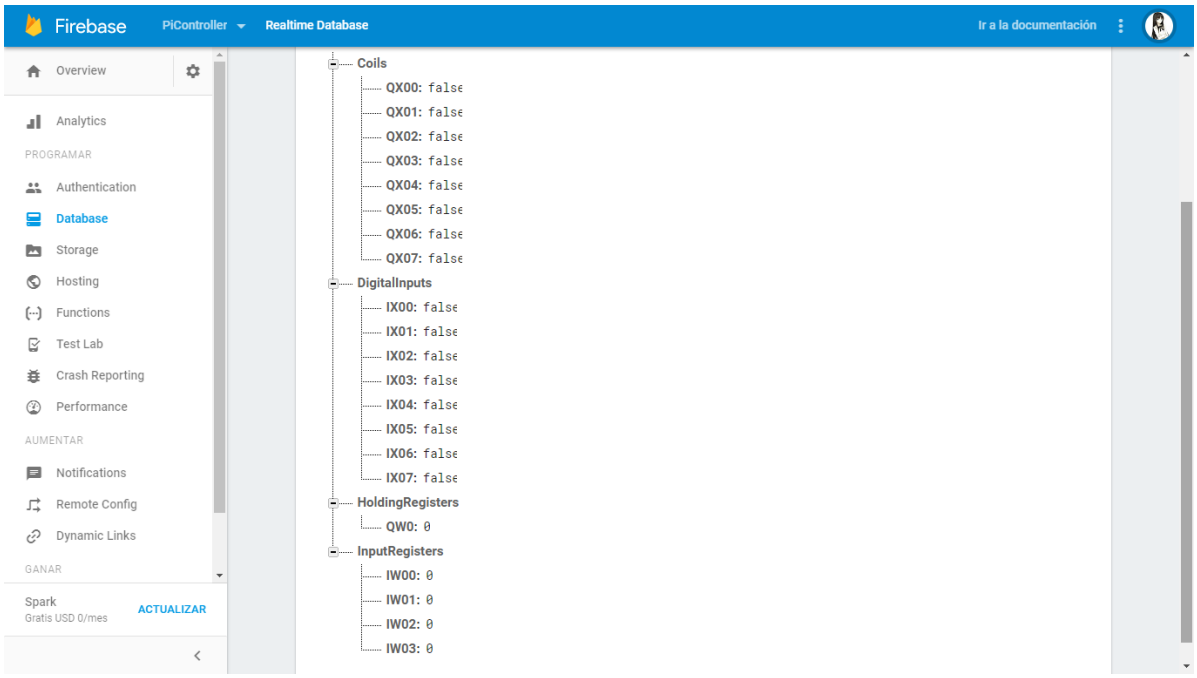
Figura 41. *Consola de Firebase*



Fuente: Elaboración Propia

Una vez generado el proyecto, se despliega un menú con varias operaciones, tales como configuraciones, análisis en tiempo real, almacenamiento, reporte de rendimiento, entre otras. Particularmente se hace uso de la base de datos de tiempo real, que se elabora como un árbol dividido en categorías, que corresponden a las entradas digitales y analógicas (*DigitalInputs, InputRegisters*) y las salidas digitales y analógicas (*Coils, HoldingRegisters*). Cada casilla almacena el valor o estado que le corresponde, por lo que solamente se emplearán valores booleanos o enteros. Por defecto, los valores digitales se encuentran en estado inactivo o *false*, y los valores analógicos en 0, como se muestra en la figura 42.

Figura 42. *Base de Datos en tiempo real de OpenPi Controller*



Fuente: Elaboración Propia

Es importante tener en cuenta que el servicio de Firebase requiere establecer una serie de reglas que indican los métodos de acceso y si los usuarios tienen permiso de lectura y escritura. Debido a que el desarrollo del OpenPi Controller se encuentra aún en fase experimental, se cambiaron a permiso libre, para que sea posible enviar datos y recibirlos desde la base datos, sin necesidad de autenticarse previamente con una cuenta de Google (Ver figura 43).

Cabe resaltar que, en fases posteriores de desarrollo del dispositivo se realizarán ajustes a estas políticas, puesto que es un aspecto esencial de seguridad a la hora de operar todo aquello que involucre IoT.

Figura 43. *Políticas de Acceso a la base de datos*

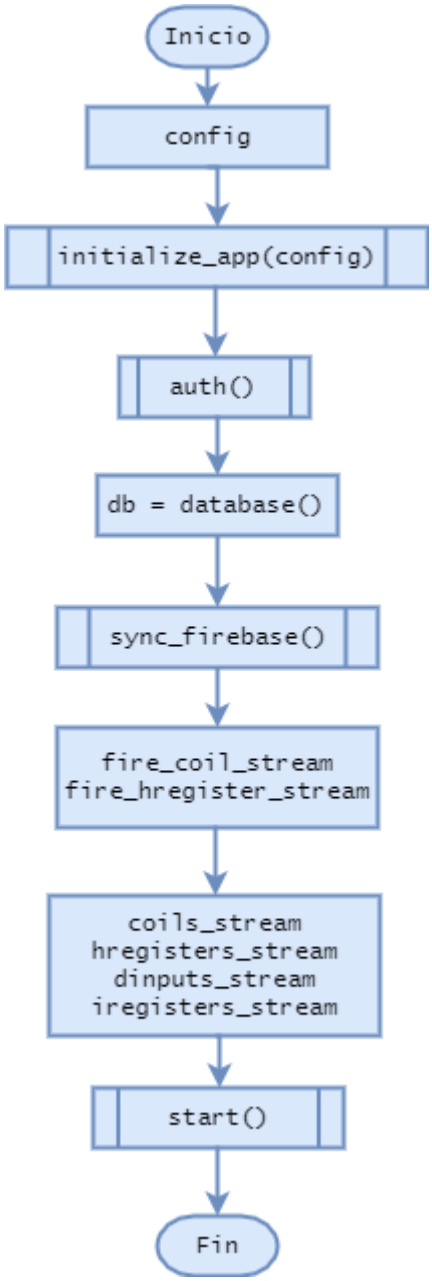
Fuente: Elaboración Propia

Ahora bien, las dos tareas principales que se requieren para conseguir un enlace completo son:

- Detección de cambios de estado en el PLC, y el envío de estos eventos hacia Firebase.
- Detección de cambios de estado en la base de datos de Firebase y su actualización hacia el PLC, mediante órdenes de escritura Modbus. Estos cambios pueden provenir de la interacción del usuario con la App móvil o la interfaz de escritorio.

Basados en *Pyrebase* (disponible en https://github.com/thisbejim/Pyrebase), una colección de funciones de Python que facilitan la interacción con Firebase, y una clase de autoría propia que lleva el nombre de **ModbusStream**, se creó el script **RemotePiCloud.py**, cuya estructura se resume en el diagrama de la figura 44.

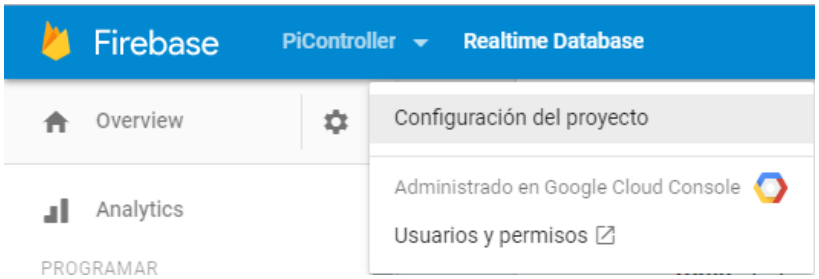
Figura 44. *Diagrama de enlace de OpenPi Controller con Firebase*



Fuente: Elaboración Propia

La variable **config** es un arreglo en el que se debe incluir una clave conocida como *APIkey*, que es generada por la plataforma. Para obtenerla, se debe acceder a la configuración del proyecto, en el ítem que muestra la figura 45.

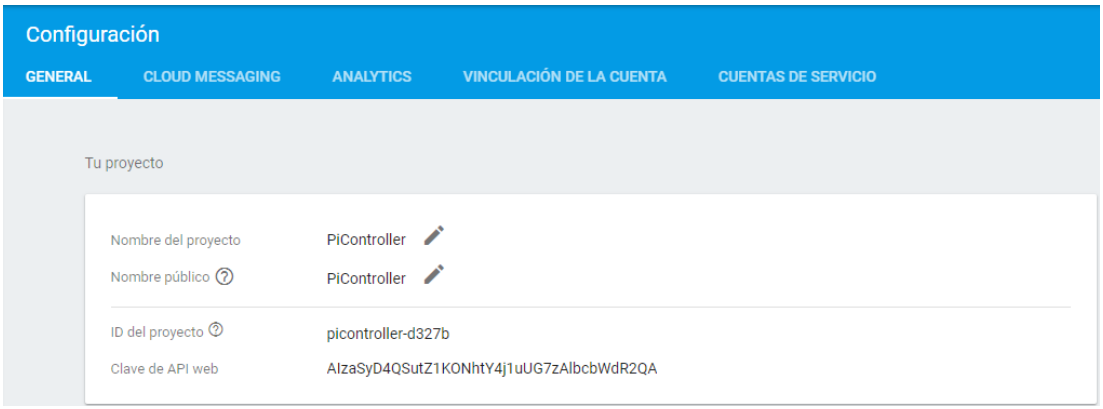
Figura 45. *Ítem de configuración de Firebase*



Fuente: Elaboración Propia

Como se puede apreciar en la figura 46, la *Clave de la API* es una cadena de números y caracteres que cumple la función de identificar al programador, al usuario y al proyecto en Firebase.

Figura 46. *Clave API del proyecto Picontroller en Firebase*



Fuente: Elaboración Propia

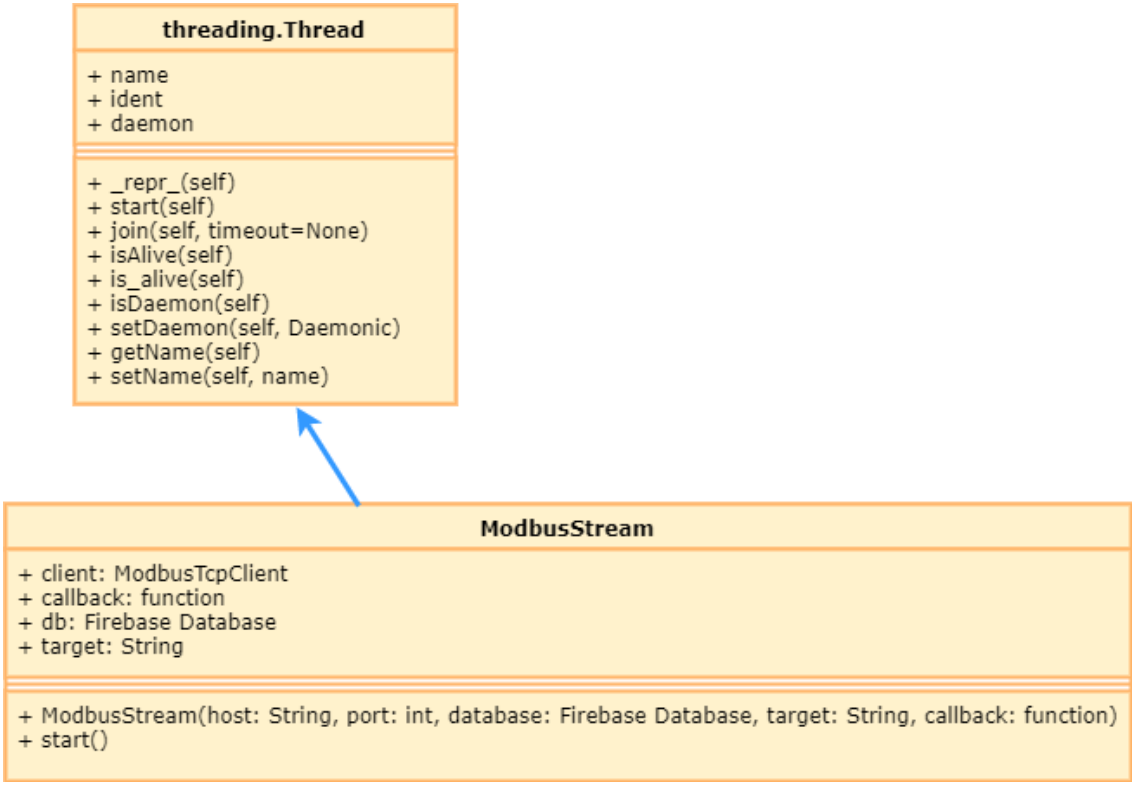
Una vez completo este paso, la información es previamente serializada gracias a **initialize_app()**, es decir, que es codificada para su envío a Firebase, y luego se autentica mediante el llamado a la función **auth()**. En el siguiente bloque se crea una instancia de la conexión hacia la base de datos, que se almacena en la variable **db**, y permitirá realizar operaciones de lectura y escritura en tiempo real.

Luego de este proceso, se envía el resultado de una sola lectura del OpenPi Controller hacia la nube, puesto que con ello asegura que el estado actual se sincronice correctamente. Seguidamente, se instancian dos objetos tipo **Stream**, provenientes de la librería *Pyrebase*, cuya función es de crear un flujo de datos desde la nube hacia el OpenPi Controller en cuanto detecte un cambio en cualquiera de las variables del árbol que representa la base de datos, mostrada en la figura 42.

Sin embargo, aunque se dispone de una librería para la lectura de estados en el PLC vía solicitudes ModbusTCP (https://github.com/uzumaxy/pymodbus3), ésta no puede detectar específicamente los cambios respecto a una lectura anterior, por lo que se diseñó una clase que se comporta de manera análoga a la clase Stream de *Pyrebase* (Ver figura 47). Teniendo en cuenta que la supervisión es una tarea que debe realizarse todo el tiempo, y de manera simultánea, se heredó la composición de la clase Thread, proveniente de la librería estándar de python. Un *thread* o hilo, permite la ejecución de múltiples tareas, que es justamente lo que se requiere para los 4 tipos de registros del PLC.

La clase **ModbusStream** recibe en su constructor 4 objetos como parámetros, estos son, un tipo ***String*** que representa la dirección IP donde se encuentra el OpenPi Controller, un tipo ***int*** que indica el puerto donde está activo el servicio esclavo del Modbus, un tipo ***Firebase Database*** que es la conexión a la base de datos previamente instanciada, otro tipo ***String*** que indica el tipo de registro que se desea supervisar (*coils*, *digital_inputs*, *holding_registers o input_registers*) y por último un tipo ***function*** que corresponde al callback que se llama al detectar los cambios.

Figura 47. *Diagrama de Clase ModbusStream*

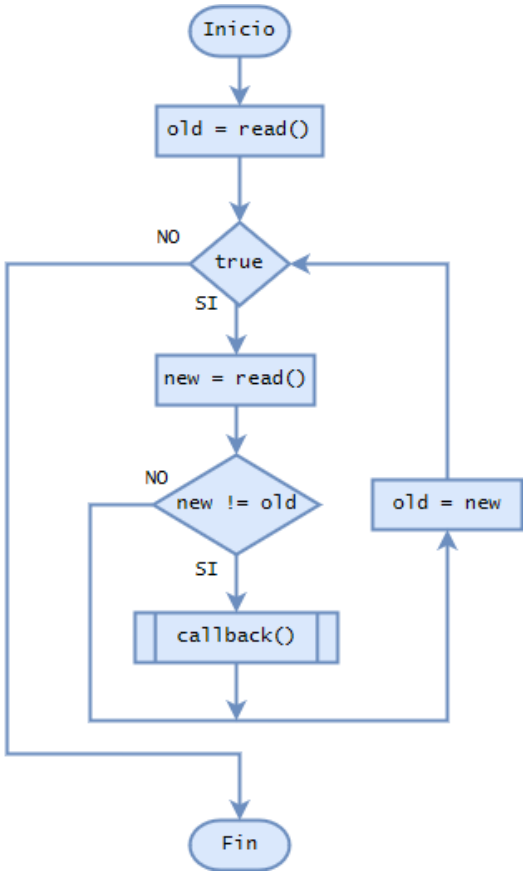


Fuente: Elaboración Propia

Ahora bien, el algoritmo para llevar a cabo la tarea de monitoreo es relativamente sencillo, y se ilustra en la figura 48. Se realiza en primer lugar, la lectura del estado inicial y se asigna este valor a la variable ***old***, y luego se inicia un ciclo indeterminado que asignará las siguientes lecturas a la variable ***new*** y actualizará ambos valores al final de cada iteración.

Como se puede apreciar, justo antes de realizar otra lectura, se comparan los valores de ***old*** y ***new***, de manera que cuando estos se diferencien en algo, quiere decir que se presentó un cambio de estado y se hará un llamado a la función callback que se pasó como parámetro al crear el *stream*, que en este caso particular, envía los datos a la nube para realizar la actualización en la base de datos. Para que funcione de manera efectiva, se creó una tarea programada en la Raspberry con la utilidad *crontab*, para que corra el script en cada inicio, dejando de esa manera, completamente automatizado todo el proceso de sincronización descrito anteriormente.

Figura 48. *Diagrama de Detección de Cambio de estados del OpenPi Controller*

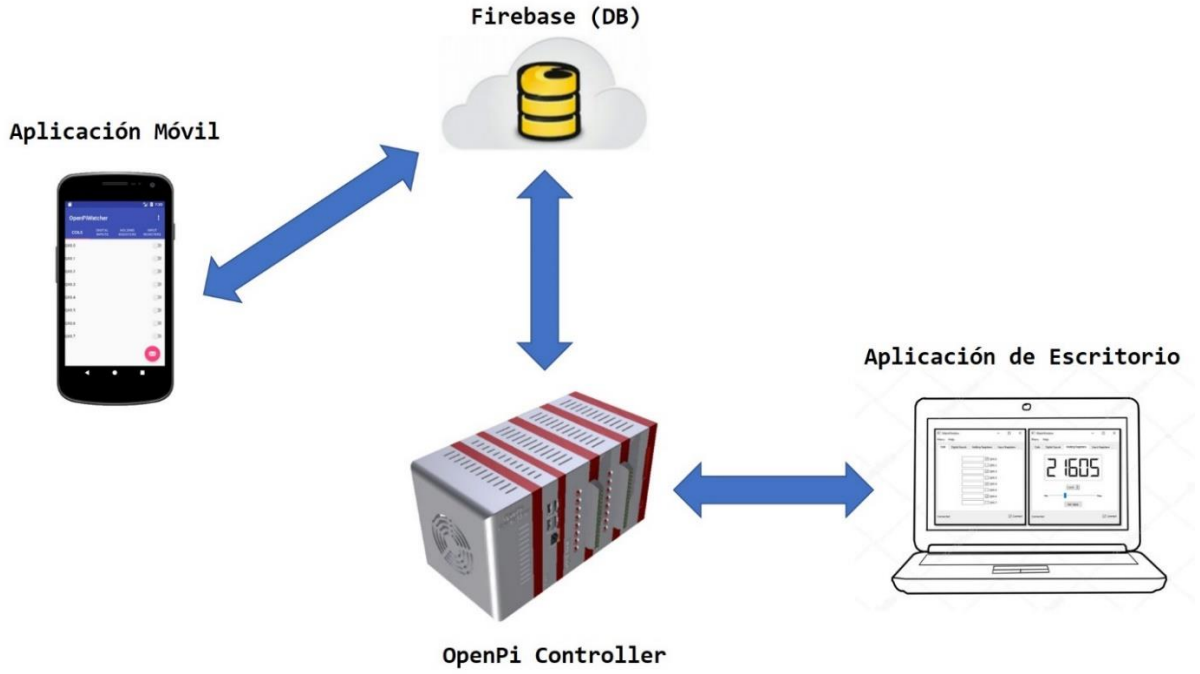


Fuente: Elaboración Propia

Es importante mencionar que se incluyó en el script una forma sencilla y clara de visualizar cada cambio que ocurre tanto en el dispositivo, como en Firebase.

2.2.5 Diseño de la Aplicación de Control y Supervisión Remota: OpenPi Watcher

Figura 49. *Diagrama de operación de la aplicación OpenPi Watcher*



Fuente: Elaboración Propia

En la figura 49 se aprecia el sistema de operación de la aplicación *OpenPi Watcher* en sus dos versiones (móvil y escritorio), diseñada para la supervisión y control del dispositivo. A continuación, se explicará el diseño y funcionalidad de cada una de las versiones de la aplicación.

2.2.5.1 Versión Móvil

Con el fin de supervisar y modificar de manera remota el estado de los registros del OpenPi Controller, se diseñó un aplicativo móvil basado en Firebase utilizando Android Studio, que puede usarse desde cualquier lugar del mundo, siempre que tenga conexión a internet.

La aplicación se compone de cuatro pestañas. Cada una de ellas corresponde a uno de los tipos de registros que posee un PLC. Como se muestra en la tabla 12 dentro de las funciones del protocolo Modbus TCP, solamente las salidas digitales (*coils*) y las salidas analógicas (*holding_registers)* permiten operaciones de escritura, por lo que solo estas dos pestañas permitirán órdenes remotas por parte del usuario, y las otras dos son exclusivamente de supervisión (*digital_inputs / input_registers*).

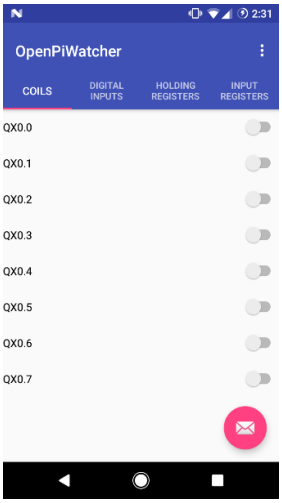
Tabla 12. *Funciones y Códigos de Operación de Modbus TCP*

Función	Código	Descripción
Read Coils	1	Leer estado de salidas discretas.
Read Discrete Inputs	2	Leer estado de entradas discretas.
Read Holding Register	3	Leer valores de registros “Holding”.
Read Input Register	4	Leer valores de registro de entrada.
Write Single Coil	5	Permite modificar el valor de una sola salida discreta.
Write Register	6	Escribe un valor en un registro “Holding”.
Write Multiple Coils	15	Permite modificar el valor de múltiples salidas discretas al tiempo
Write Multiple Registers	16	Escribe múltiples registros “Holding” al mismo tiempo.

Fuente: ModbusOrg (2006a). Modbus application protocol specification v1.1b. Technical report, The Modbus Organization.

La primera pestaña corresponde a las salidas digitales del OpenPi Controller (Ver figura 50), cuyo estado de cada registro (verdadero o falso) se representa como un interruptor activado o desactivado. La aplicación se encarga de enviar periódicamente paquetes con los cambios que realiza el usuario al pulsar sobre los interruptores, y los aplica directamente al PLC mediante el enlace a la nube previamente mencionado.

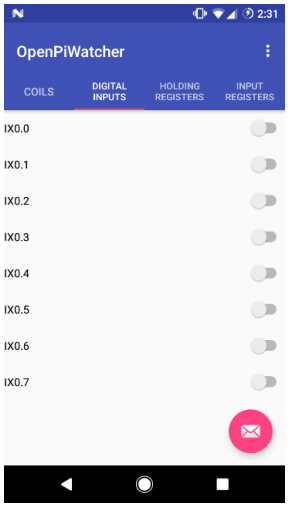
Figura 50. *Pestaña Coils*



Fuente: Elaboración Propia

De manera similar, la segunda pestaña se actualiza de manera constante con los valores que le proporciona Firebase del estado en que se encuentra cada uno de los registros de las entradas digitales (Ver figura 51). Como se mencionó anteriormente, no es posible interactuar con los valores de estos registros.

Figura 51. *Pestaña Digital Inputs*



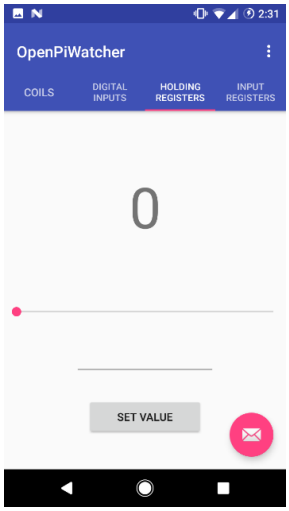
Fuente: Elaboración Propia

La tercera pestaña permite establecer el valor del registro para la salida analógica del OpenPi. Este registro es representado por un número entero de 16 bits sin signo, alcanzando un rango de valores discreto que va desde 0 hasta 65535.

$$V_{OUT} = 10 \times \frac{Valor}{65535}$$

El voltaje de salida del PLC será proporcional al valor que el usuario seleccione mediante el uso de una barra deslizante o el campo de texto numérico que se proporciona, como se observa e la figura 52.

Figura 52. *Pestaña Holding Registers*



Fuente: Elaboración Propia

Por último, la cuarta pestaña cumple la tarea de indicar como un valor porcentual la lectura para cada registro de las entradas analógicas, el valor del voltaje que se ve reflejado, siendo 10 Voltios el valor máximo de entrada (100%). Es importante mencionar que el conversor Analógico/Digital del OpenPi posee una resolución de 15 bits, lo que permite alcanzar valores discretos desde 0 hasta 32767. La fórmula para obtener el valor analógico de cada entrada se muestra es la siguiente.

$$V_{IN} = 10 \times \frac{LecturaADC}{32767}$$

Como se puede apreciar en la figura 53, ésta pestaña usa cuatro barras de progreso para indicar visualmente los cambios que ocurren.

Figura 53. *Pestaña Input Registers*

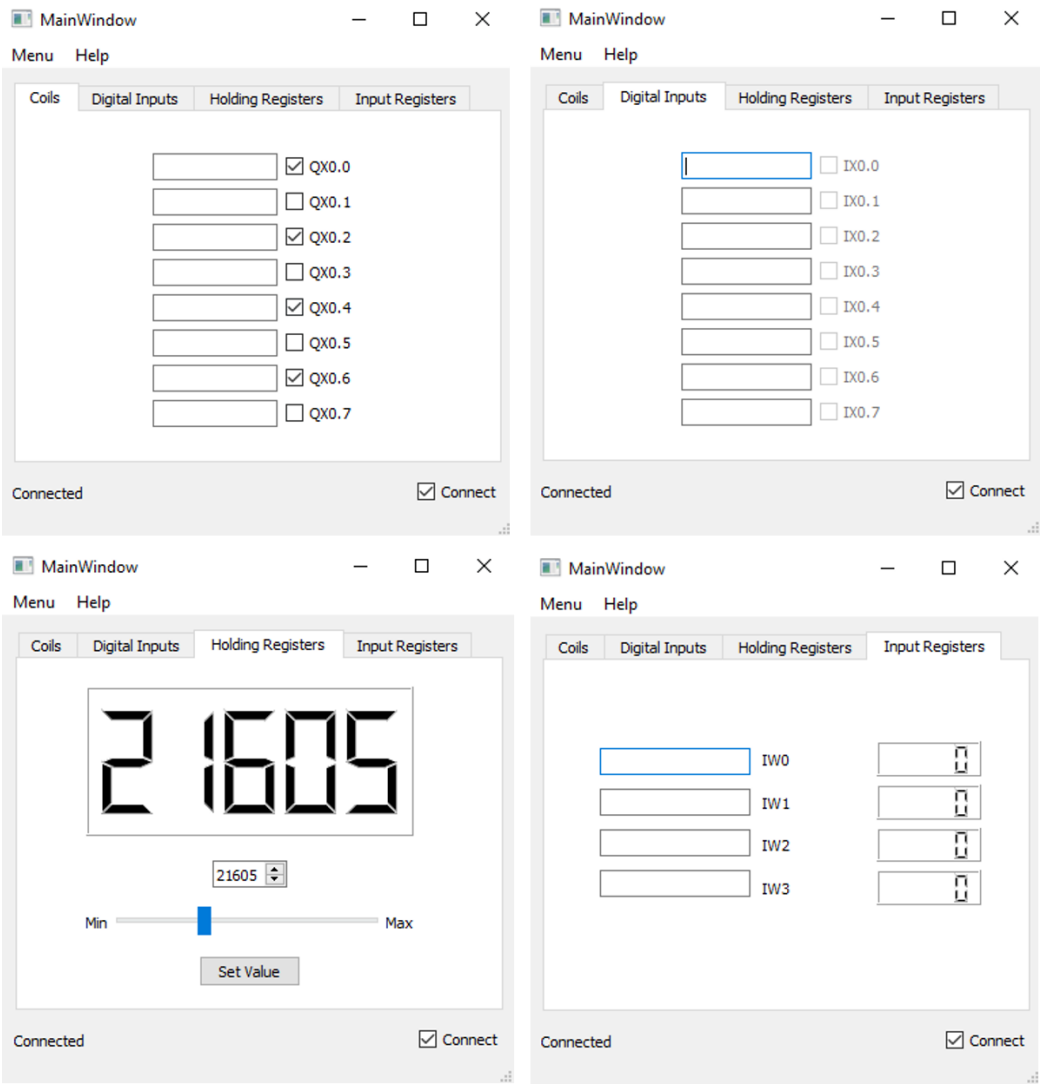
Fuente: Elaboración Propia

2.2.5.2 Versión de Escritorio

Debido a que ocasionalmente se puede presentar una falla o no exista conexión a internet, se desarrolló una aplicación de escritorio que tiene las mismas capacidades de la aplicación móvil, con la diferencia de que está diseñada para que funcione sobre red local y sin la intervención de Firebase para el envío y recepción de datos.

La aplicación está escrita en Python, se basa en pymodbus3 para la confección de paquetes Modbus y posee una sencilla interfaz en Qt5, que se divide también en cuatro pestañas con las funciones requeridas para la supervisión del Open Pi Controller (Ver figura 54).

Figura 54. *Pestañas de la Aplicación versión escritorio*



Fuente: Elaboración Propia

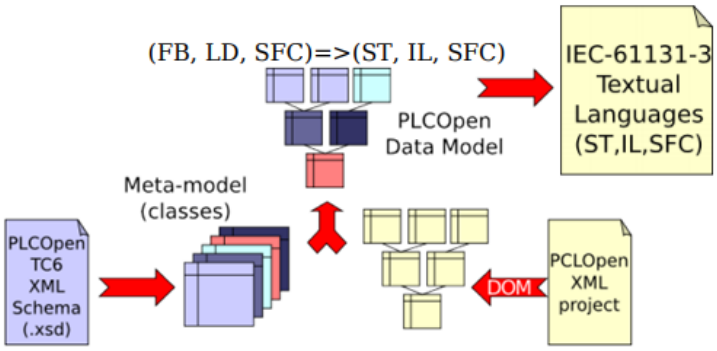
El código fuente de la aplicación OpenPi Watcher en sus dos diferentes versiones (móvil y escritorio) se encuentra disponible en el siguiente enlace: *https://github.com/Legacier/OpenPiWatcher.*

3. PLCOPEN EDITOR: PROGRAMACIÓN DEL OPENPI CONTROLLER

Para la programación del prototipo de PLC desarrollado durante este proyecto se utilizó el **PLCopen Editor**, un software que permite escribir programas o rutinas de PLC de acuerdo con la norma IEC-61131-3 y conforme a PLCopen XML. Fue creado principalmente por Edouard Tisserant y Laurent Bessard para el proyecto ***Beremiz***[17].

El PLCopen Editor se encarga de guardar y cargar proyectos de controladores lógicos programables de acuerdo con el esquema PLCopen TC6-XML, este proceso de funcionamiento se esquematiza en la figura 55.

Figura 55. *Modelo de datos de PLCopen Editor*



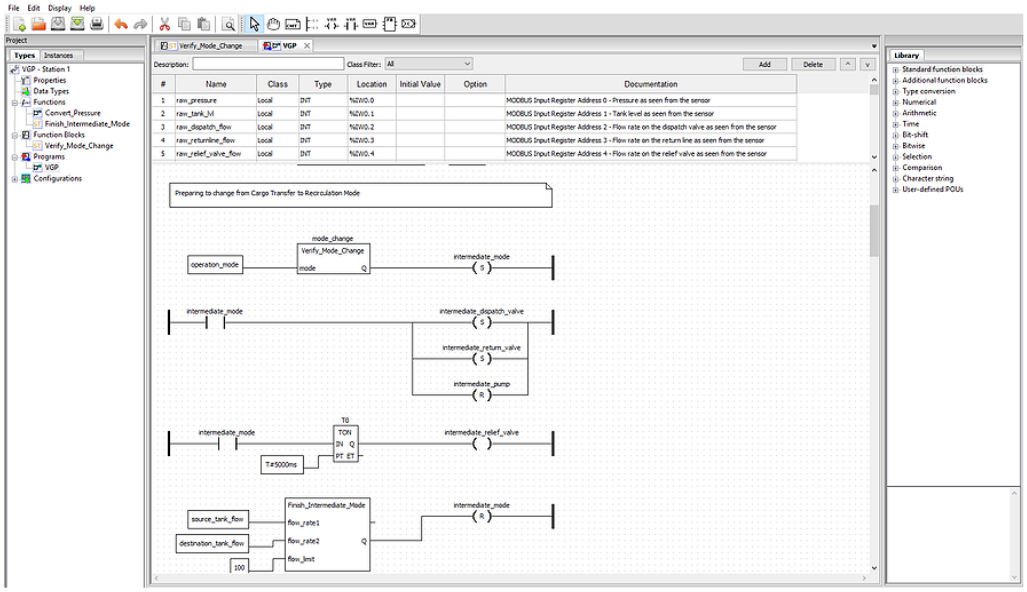
Fuente: Tomada de Beremiz User Manual
Disponible en http://www.beremiz.org/beremiz_user_manual_lolitech.pdf

En primer lugar, el archivo *oficial.xds* es usado en el arranque para crear una especie de meta-modelo, el cual es combinado con los objetos dentro de PLCopen (*proyectos XML*) para obtener el modelo de datos del PLCopen Editor; finalmente el editor incorpora un filtro de exportación que convierte los lenguajes gráficos a sus equivalentes en forma textual, que corresponderá a la información que será transmitida o cargada al controlador.

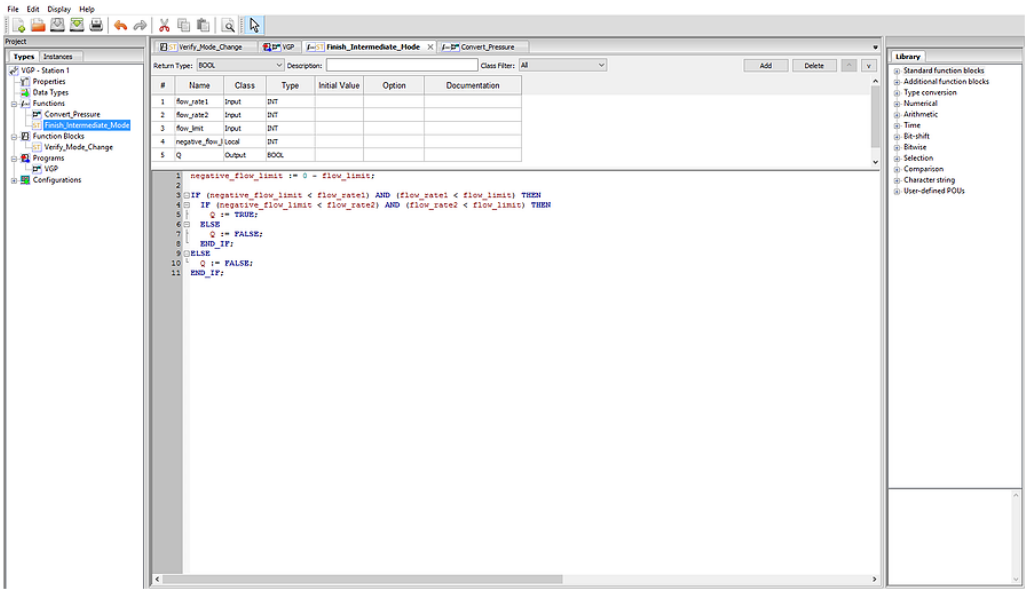
Estos programas en forma textual (ST, IL, SFC) son los que el software de OpenPi Controller será capaz de ejecutar, los cuales son cargados a través de la interfaz web de OpenPLC. El PLCopen Editor puede generar también programas en los demás lenguajes dentro del estándar IEC-61131-3. En la figura 56 se muestran algunos de los lenguajes que soporta el editor.

[17] Beremiz (2014). Beremiz free software for automation. Disponible en: http://www.beremiz.org/doc.html.en

Figura 56. *Lenguajes de Programación PLCopen Editor*



(a)



(b)

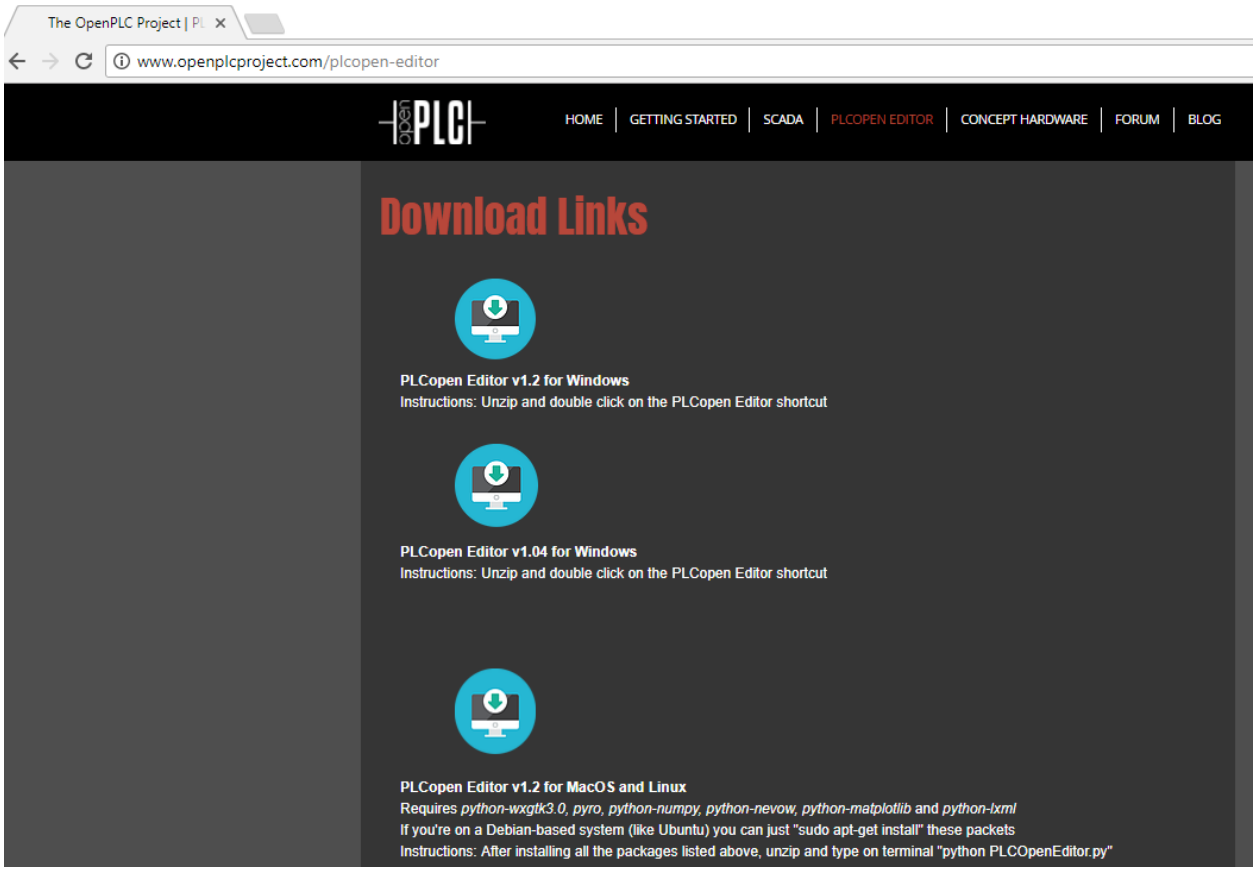
(a)LD **(b)**ST

Fuente: Elaboración Propia

3.1 DESCARGA E INSTALACIÓN

El PLCopen Editor se puede descargar del sitio oficial del proyecto OpenPLC en la pestaña PLCOPEN EDITOR (http://www.openplcproject.com/plcopen-editor). Aquí se encontrarán los diferentes enlaces de descarga para cada uno de los sistemas operativos (Windows, Linux y MacOS), como se aprecia en la figura 57.

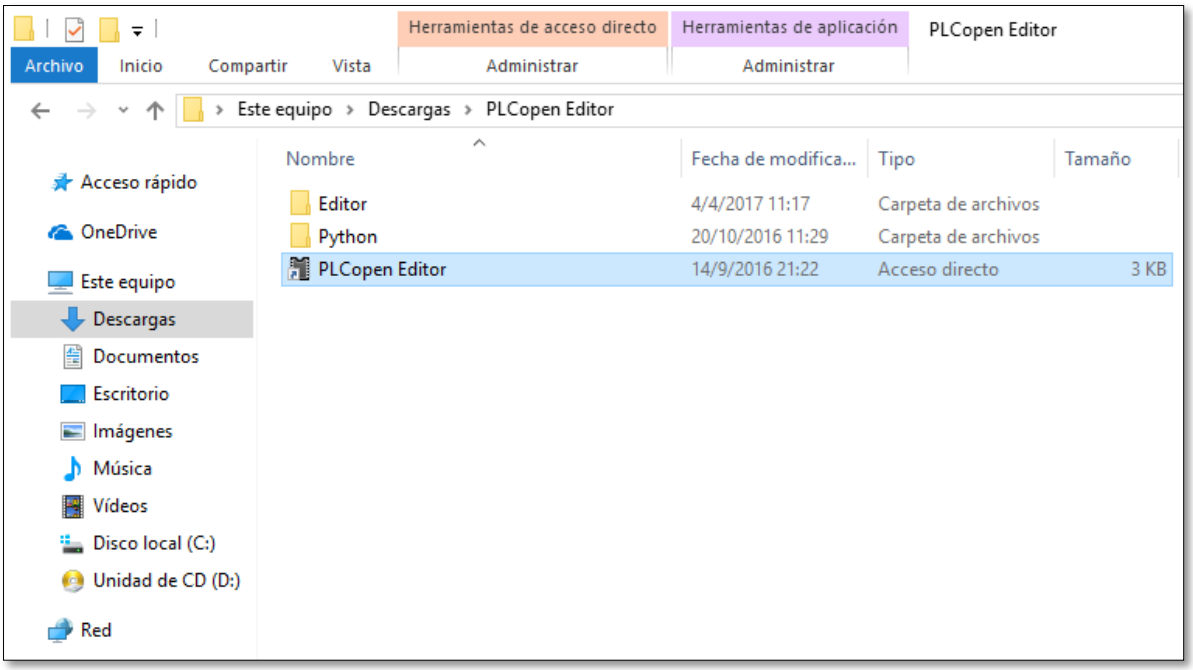
Figura 57. *Descarga PLCopen Editor*



Fuente: Elaboración Propia

Para el caso de Windows solo basta con dirigirse al enlace correspondiente (*PLCopen Editor for Windows*) descargar el archivo comprimido y ejecutar el acceso directo de la aplicación (Ver figura 58).

Figura 58. *Ejecución PLCopen Editor*



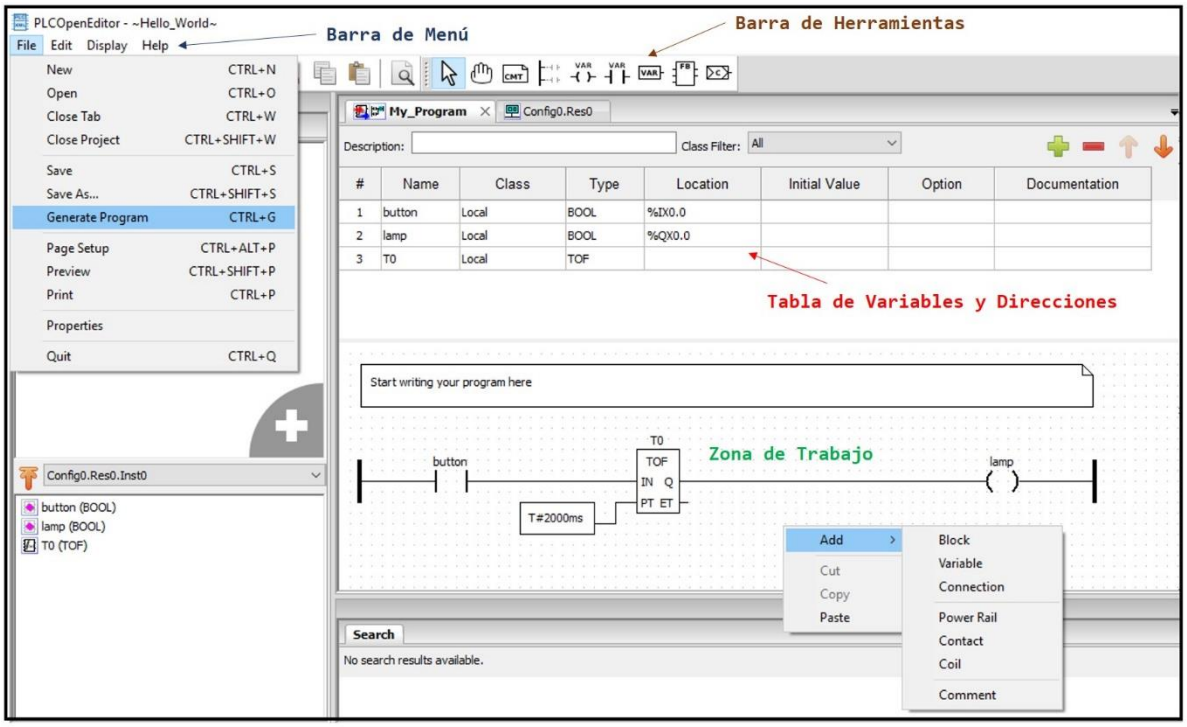
Fuente: Elaboración Propia

Por el contrario, para Linux y MacOS mediante el gestor de paquetes se instalan las dependencias indicadas junto al enlace de descarga (*PLCopen Editor for MacOS and Linux*). Luego, se descomprime el archivo descargado y se ejecuta en la terminal la siguiente orden "*python PLCOpenEditor.py*".

3.2 INSTRUCCIONES DE USO

Con la intención de entender un poco el funcionamiento del editor, se hará a continuación una revisión rápida de los principales elementos que componen su entorno de programación; para este caso se decidió utilizar el lenguaje grafico de escalera (LD), ya que este es por defecto empleado para la programación de un PLC normalmente. En la figura 59 se observan algunas de la barras y herramientas principales que se utilizan para el diseño de programas en este lenguaje dentro del editor.

Figura 59. *Ventana Principal de PLCopen Editor*



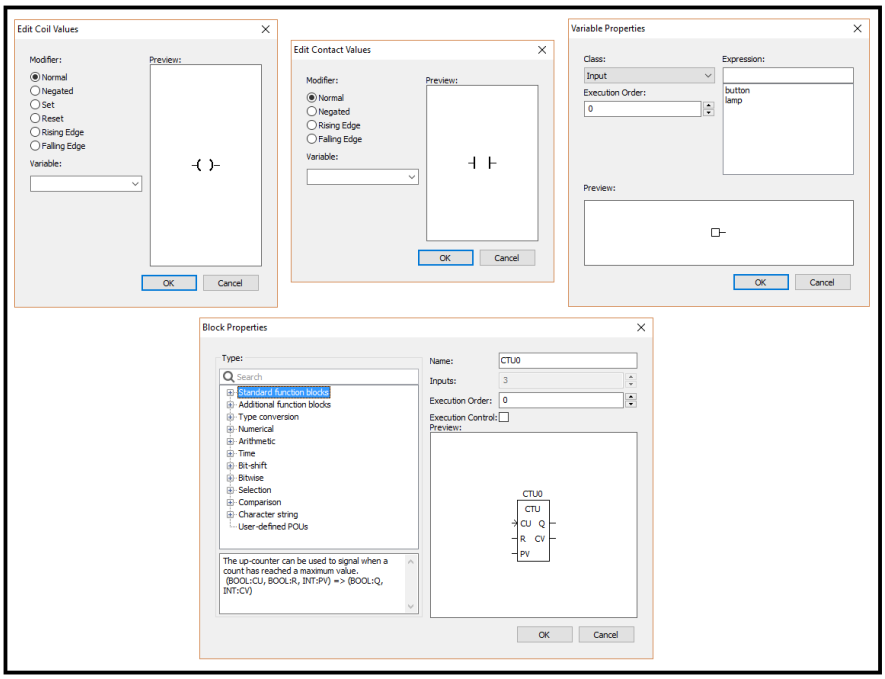
Fuente: Elaboración Propia

La ***barra de herramientas*** cuenta con opciones como *nuevo*, *abrir* y *guardar* las cuales se utilizan para el diseño de nuevos proyectos, además del uso de proyectos recientes; estas también vienen por defecto en la pestaña *archivo* de la ***barra de menú*** del programa. Por otra parte, la barra de herramientas también tiene elementos como *cortar, copiar y pegar* que se emplean dentro de la ***zona de trabajo*** durante el diseño de proyectos.

Ya que el editor continúa en fase de desarrollo experimental, es recomendable iniciar un nuevo proyecto a partir de un archivo estándar conocido como *Hello_World.xml*, que contiene la plantilla de un programa elemental que se puede descargar del sitio oficial de OpenPLC en el siguiente enlace *http://www.openplcproject.com/getting-started-rpi*; sobrescribiendo este programa, el usuario podrá crear proyectos desde cero utilizando los diferentes elementos de diseño que se encuentran en la sección correspondiente de la barra de herramientas. Esta sección está conformada por elementos como *Coils*, *contactos*, *variables*, *bloques de funciones*, entre otros; de igual forma, estos componentes pueden ser llamados directamente desde la zona de diseño haciendo clic derecho sobre esta y usando la opción *Add* que desplegara una lista con cada uno de estos elementos.

Ahora bien, cada uno de los diferentes elementos nombrados anteriormente cuenta con preferencias y parámetros de diseño, que el usuario será capaz de modificar y editar según las características o necesidades del proyecto que esté desarrollando. En la figura 60 se encuentran las ventanas que se despliegan cuando se accede a alguno de los elementos de diseño; como se puede ver, parámetros como contactos y Coils solo permiten modificar sus valores seleccionando estados ya predeterminados (*normal, negado, flanco ascendente, flanco descendente*), por el contrario, las variables y los bloques de funciones contienen propiedades específicas que se podrán editar de acuerdo al usuario.

Figura 60. *Elementos de diseño de Diagrama Ladder PLCopen Editor*



Fuente: Elaboración Propia

Luego de haber finalizado el diseño del diagrama de escalera, el siguiente paso será especificar dentro de la ***tabla de variables y direcciones***, los nombres de los elementos de diseño que representaran los componentes físicos (sensores, actuadores, dispositivos de control) con los cuales trabajara directamente el PLC, además de los tipos de datos estándar que manejara cada uno de ellos y finalmente sus localizaciones para su respectiva conexión física. En la tabla 13 se resumen las direcciones de entradas y salidas del OpenPLC, que se asignan de igual forma a las direcciones que maneja Modbus y OpenPi Controller; además, se especifican los tipos de registro, valores, usos y accesos disponibles.

Tabla 13. *Dirección de Registros OpenPi Controller*

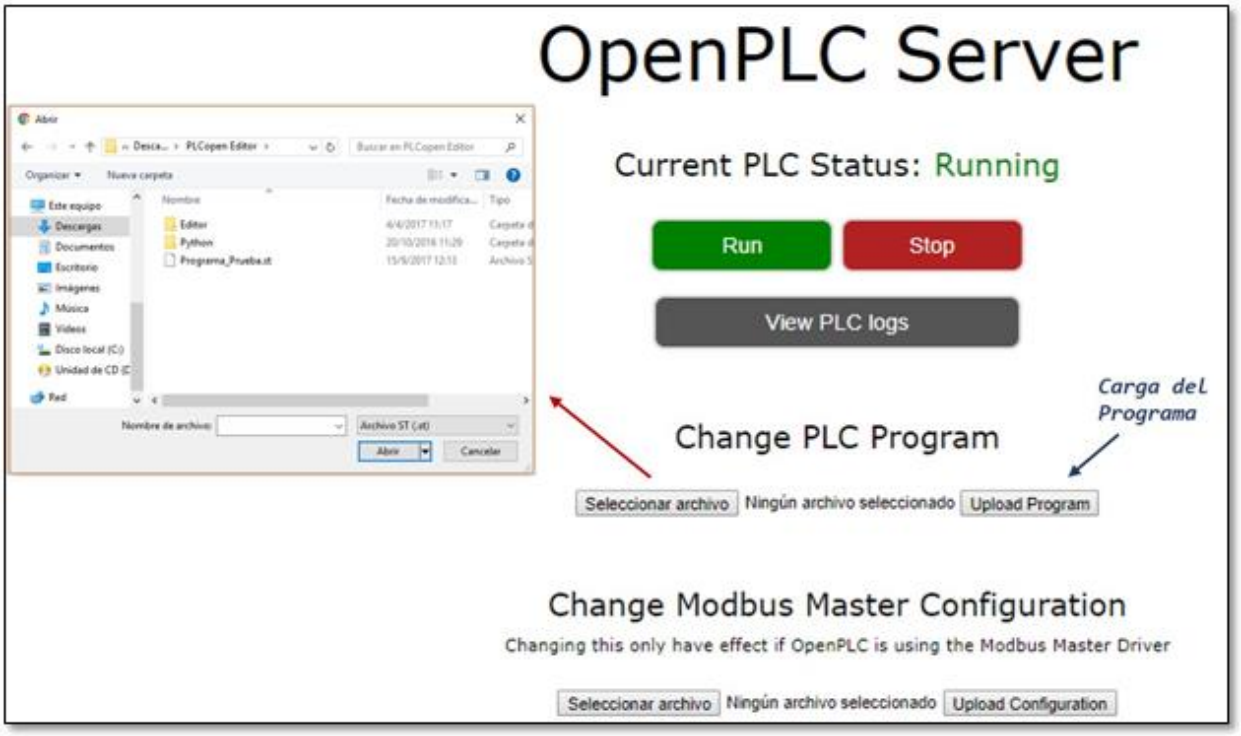
Tipo de Registro	Uso	Dirección PLC	Dirección Modbus	Rango de Valores	Acceso
Coil Registers	Salidas digitales	%QX0.0 - %QX99.7	0 - 799	0 o 1	Lectura y Escritura
Discrte Input Registers	Entradas digitales	%IX0.0 - %IX99.7	0 - 799	0 o 1	Lectura
Input Registers	Entradas analógicas	%IW0 - %IW99	0 - 1023	0 a 65535	Lectura
Holding Registers	Salidas analógicas	%QW0 - %QW99	0 - 1023	0 a 65535	Lectura y Escritura

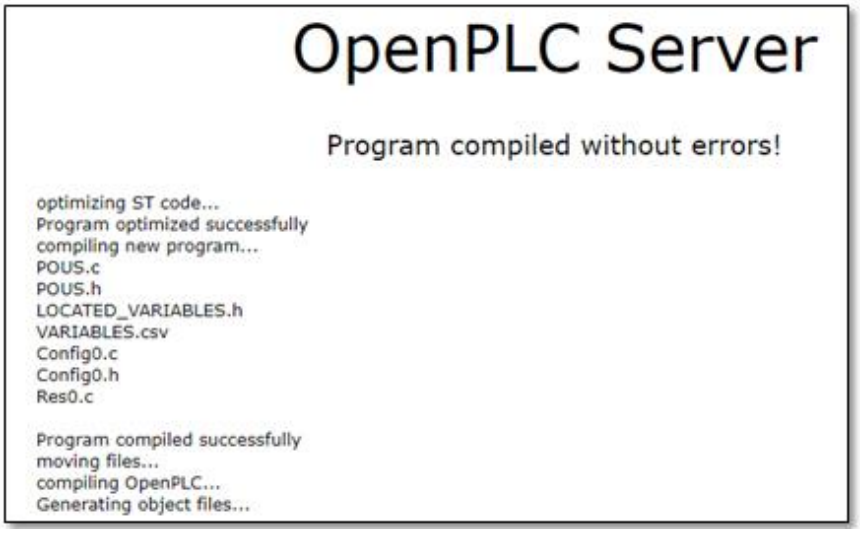
Fuente: http://www.openplcproject.com/scada

Por último, para compilar el diagrama de escalera en un archivo ST, se empleará la opción *Generar Programa* dentro del menú archivo; y a continuación, se cargará en el OpenPi Controller utilizando la interfaz web de OpenPLC.

Esta interfaz cuenta con un botón "*Seleccionar archivo*" que permite al usuario elegir el programa generado en el editor previamente, y a través del botón "*Upload Program*" iniciar la carga y compilación de este mismo al OpenPi Controller. En la figura 61 se observa este procedimiento y el aviso que confirmara el proceso exitoso de compilación y carga del programa.

Figura 61. *Carga y Compilación de un Programa al OpenPi Controller*





Fuente: Elaboración Propia

4. RESULTADOS

Para evaluar el OpenPi Controller como un PLC real, se realizó un test de comparación para probar su rendimiento frente a un controlador de una gama similar, disponible en el mercado. Esto se lleva a cabo empleando un modelo a escala de un proceso industrial, que consiste en 3 etapas (Ver figura 62):

Figura 62. *Modelo a escala de un sistema de tanques*

Fuente: Elaboración Propia

- Después de presionar un pulsador, inicia el bombeo de agua desde un tanque de reserva hacia el principal, durante 8 segundos. Se controla una mini-bomba DC de 6V.
- Mezclado del tanque principal durante 10 segundos. Se controla un motor DC de 5V con aspas sumergidas en el agua.
- Drenado de agua del tanque principal durante 5 segundos. Se controla una electroválvula de 12V que permite o inhibe el flujo de líquido.

Cada una de las etapas enciende durante su ejecución un led de 24V para indicar visualmente el progreso del proceso, hasta que culmina.

El controlador que se seleccionó para llevar a cabo el test de comparación fue el PLC Micrologix850 de la marca Allen Bradley (Ver figura 63).

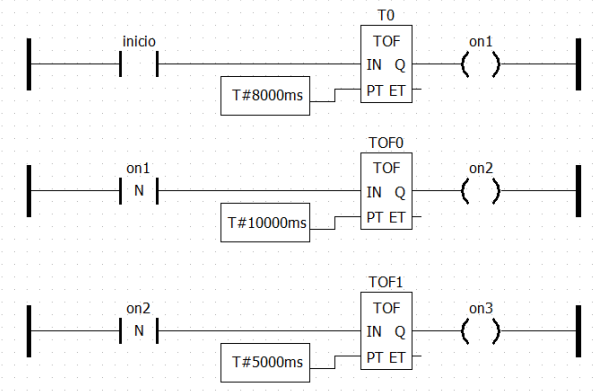
Figura 63. *PLC Micro850 Allen Bradley*

Fuente: https://www.indiamart.com/synergyautomation/allen-bradley-plc.html

Este controlador de 24 puntos con estilo ladrillo tiene funciones expandibles y puede aceptar hasta cuatro módulos de E/S de expansión, proporcionando una mayor flexibilidad y rendimiento. Cuenta con 14 entradas y 10 salidas, está especialmente diseñado para aplicaciones de máquinas autónomas de mayor tamaño, y el módulo de E/S de expansión Micro850 ofrece como valor agregado una mayor densidad y mayor precisión de E/S analógicas y digitales, según se requiera[18].

El diagrama de escalera para esta tarea fue escrito tanto en *Connected Workbench Components* (figura 64) como en *PLCOpen Editor* (figura 65) usando los mismos bloques para cada uno de ellos, teniendo como variables 3 entradas digitales y 3 salidas digitales para controlar todo el modelo a escala, además de 3 temporizadores internos.

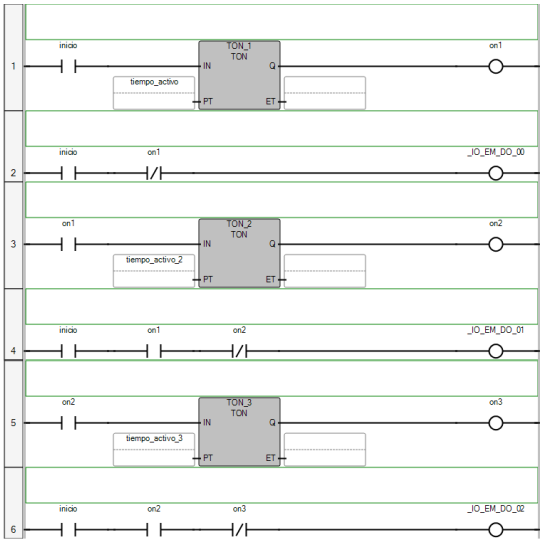
Figura 64. *Diagrama Ladder en PLCopen Editor*



Fuente: Elaboración Propia

[18] Manual de usuario Controladores Micro830 y Micro850. Disponible en: http://literature.rockwellautomation.com/idc/groups/literature/documents/um/2080-um002_-es-e.pdf

Figura 65. *Diagrama Ladder en Connected Workbench Components*



Fuente: Elaboración Propia

En las figuras 66 y 67 se muestra la estructura de las tablas de variables y direcciones para cada editor respectivamente.

Figura 66. *Tabla de variables y direcciones - Workbench*

	Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data Type	Dimension	Comment	String Size
	tiempo_activo		T#10s	N/A			TIME			
▶	inicio			N/A			BOOL			
+	TON_1		...	...	...		TON			
	on1			N/A			BOOL			
+	TON_2		...	...	...		TON			
	on2			N/A			BOOL			
	tiempo_activo_2		T#6s	N/A			TIME			
+	TON_3		...	...	...		TON			
	on3			N/A			BOOL			
	tiempo_activo_3		T#4s	N/A			TIME			

Fuente: Elaboración Propia

Figura 67. *Tabla de variables y direcciones – PLCOpen Editor*

My_Program

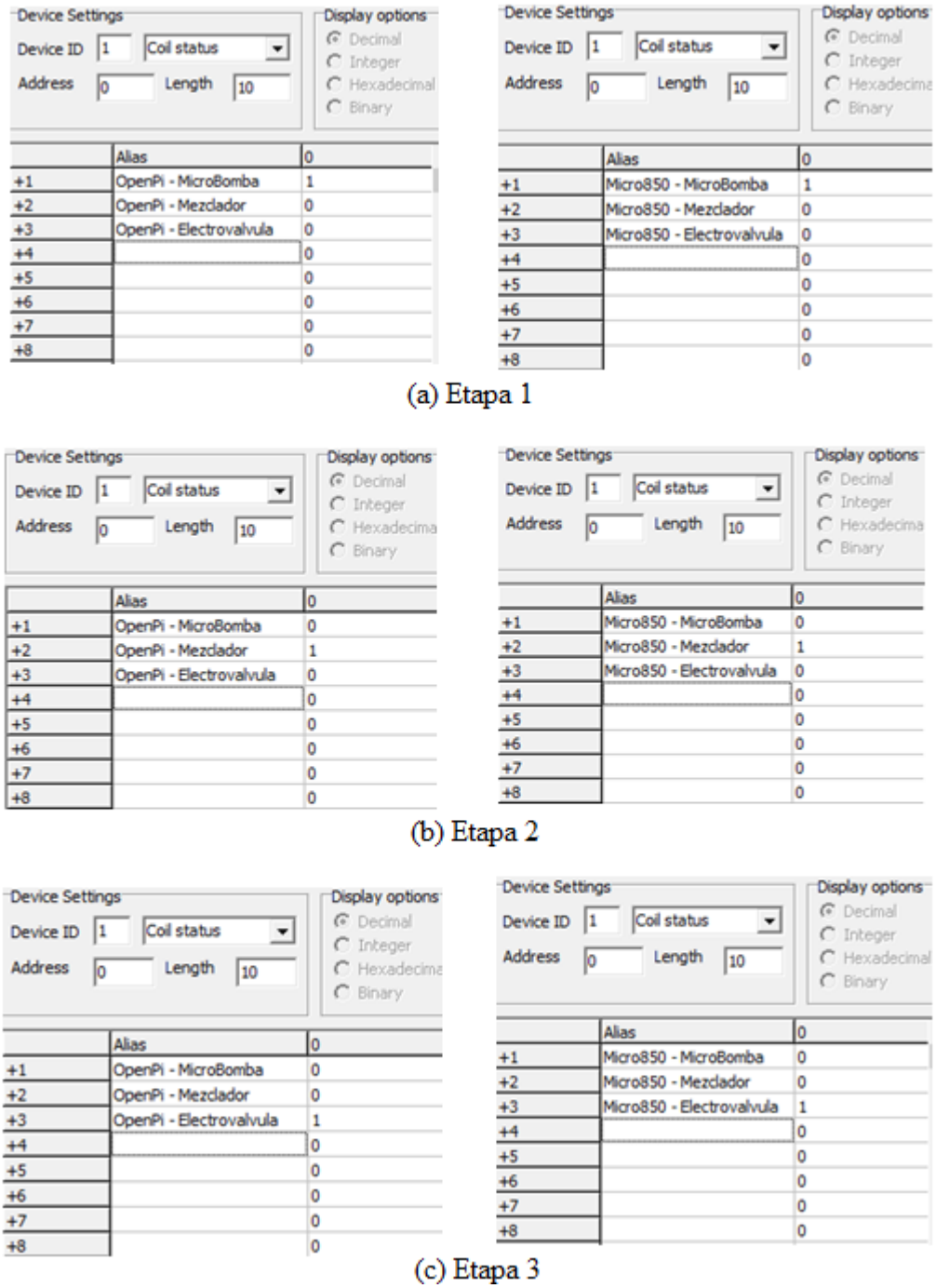
Description: Class Filter: Todo

#	Nombre	Class	Tipo	Location	Initial Value	Option	Documentation
1	inicio	Local	BOOL	%IX0.0			
2	on1	Local	BOOL	%QX0.0			
3	on2	Local	BOOL	%QX0.1			
4	on3	Local	BOOL	%QX0.2			
5	T0	Local	TOF				
6	TOF0	Local	TOF				
7	TOF1	Local	TOF				

Fuente: Elaboración Propia

Con ayuda del software Radzio, mediante solicitudes Modbus/TCP se realizaron lecturas periódicas para monitorear el estado de las salidas digitales en cada etapa, como se observa en figura 68.

Figura 68. *Estados de salidas digitales OpenPi Controller vs Micro850*



Fuente: Elaboración Propia

Al terminar la prueba, se comprobó que el OpenPi Controller realizó la tarea y presentó el mismo comportamiento del PLC Micro850, tal como se esperaba. Esto significa que la respuesta a diferentes estímulos en las entradas de cada PLC es idéntica para el problema propuesto. Es importante mencionar que ambos dispositivos hicieron uso de una fuente externa para la salida que controla el estado de la micro bomba, puesto que éste requiere aproximadamente 2A para transportar el líquido.

5. CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

- En este trabajo, se propuso un prototipo de un PLC modular, basado en herramientas de software y hardware libre. Se diseñó a partir del software OpenPLC, que emula el comportamiento de un controlador industrial, siendo capaz de ejecutar programas del estándar IEC 61131-3 y responder a solicitudes Modbus/TCP. Además, se añadió la capacidad de integrar el dispositivo con la nube, de manera que pueda ser monitoreado y controlado remotamente por cualquier interfaz HMI genérica o por medio del aplicativo móvil que se diseñó específicamente para este proyecto.

- Se comparó el rendimiento del prototipo frente a un controlador de la misma gama de la serie micro 800 de Allen Bradley. Se observó que ambos controladores responden de manera idéntica a los cambios de estados en las entradas, y sus registros también se modifican al recibir la instrucción Modbus respectiva. A su vez, se evidenció una notable reducción de los costos respecto a la adquisición de cada PLC y sus respectivos módulos.

- Debido a que el dispositivo se trata de un proyecto de código abierto y posee arquitectura modular, es posible emplear únicamente los módulos y componentes que se ajustan al requerimiento específico del usuario y no limitados a una sola marca o fabricante. Además, cuenta con toda una comunidad de desarrolladores que dan soporte frente a una eventual falla o la necesidad de añadir una nueva característica al controlador, permitiendo su constante mejora y actualización.

- El prototipo aún se encuentra en fase experimental, puesto que algunas características como la robustez y protección de la cubierta, la estructura y forma de conexión del riel, el requerimiento de ancho de banda para una operación fluida, entre otros aspectos, pueden mejorarse para obtener una experiencia de uso similar a un controlador industrial. Por tanto, se recomienda su uso con enfoque académico, para el estudio de materias relacionadas con el ámbito de la automatización.

5.2 RECOMENDACIONES

- Implementar módulos de entradas y salidas analógicas de 4-20mA, puesto que una amplia gama de sensores para la industria usa ese estándar.
- Implementar los módulos de salida digital, basados en relés de estado sólido, reduciendo con ello el tamaño y el ruido audible que producen al cambiar de estado
- Modificar la estructura del puerto de comunicación y soporte, cambiando los tipos de conectores a unos estandarizados que permitan una mayor facilidad de conexión, y le brinden robustez al dispositivo.
- Mejorar el diseño de la cubierta, de manera que cumpla con el estándar IEC-60529 y pueda tener una mayor protección contra factores externos que contribuyen al deterioro del dispositivo como los impactos mecánicos, corrosión, hongos, humedad, solventes y radiación solar.

BIBLIOGRAFÍA

BALCELLS Josep y ROMERAL José Luis, (2000). *Autómatas Programables*. Barcelona. Ed. MARCOMBO, S.A.

BEREMIZ (2014). *Beremiz free software for automation*. Disponible en: http://www.beremiz.org/doc/.

IEC (2013). *International standard IEC 61131-3 programmable controllers* – part 3: Programming languages. Technical report, International Electrotechnical Commission.

MEDINA David y ALVAREZ Evelin. *Controladores Lógicos Programables*. Departamento de Electrónica y Control. Universidad del Zulia. República Bolivariana.

MODBUSORG (2006a). *Modbus application protocol specification v1.1b*. Technical report, The Modbus Organization.

ROCKWELL AUTOMATION. *Manual de usuario Controladores Micro830 y Micro850*. Publicación 2080-UM002F (2013). Disponible en web: http://literature.rockwellautomation.com/idc/groups/literature/documents/um/2080-um002_-es-e.pdf.

PORRAS A. y MONTANERO A.P., (1992). *Autómatas Programables: fundamentos, manejo, instalación y prácticas*. Ed. Mc Graw Hill.

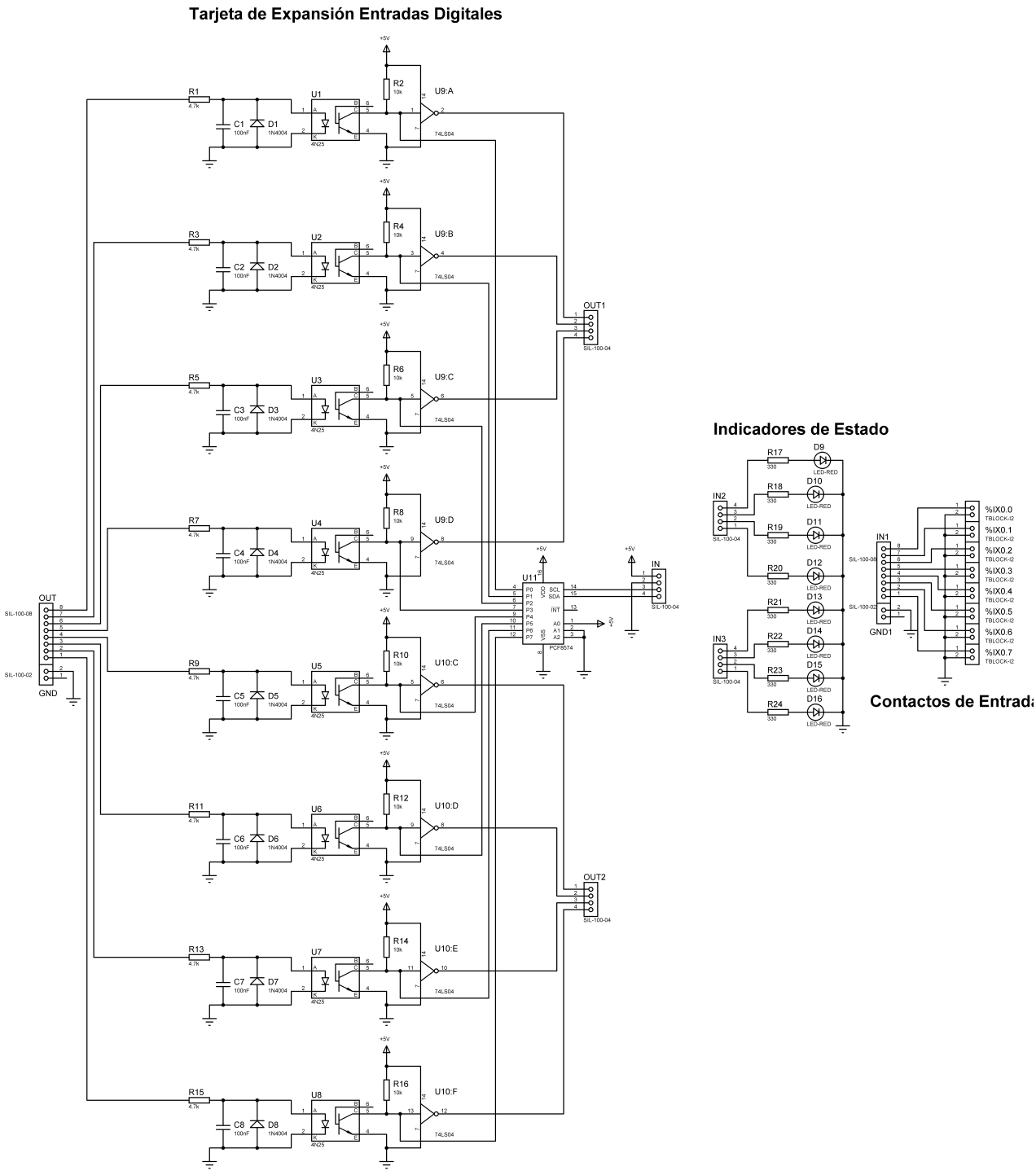
SALAZAR SERNA César Augusto y CORREA ORTIZ, Luis Carlos (2011). *Buses de campo y Protocolos en redes industriales*. Manizales (Colombia): Facultad de Ciencias e Ingeniería, Universidad de Manizales. p. 83-109. ISSN: 0123-9678.

T. R. Alves, M. Buratto, F. M. de Souza and T. V. Rodrigues, *"OpenPLC: An open source alternative to automation"* IEEE Global Humanitarian Technology Conference (GHTC 2014), San Jose, CA, 2014, pp. 585-589.

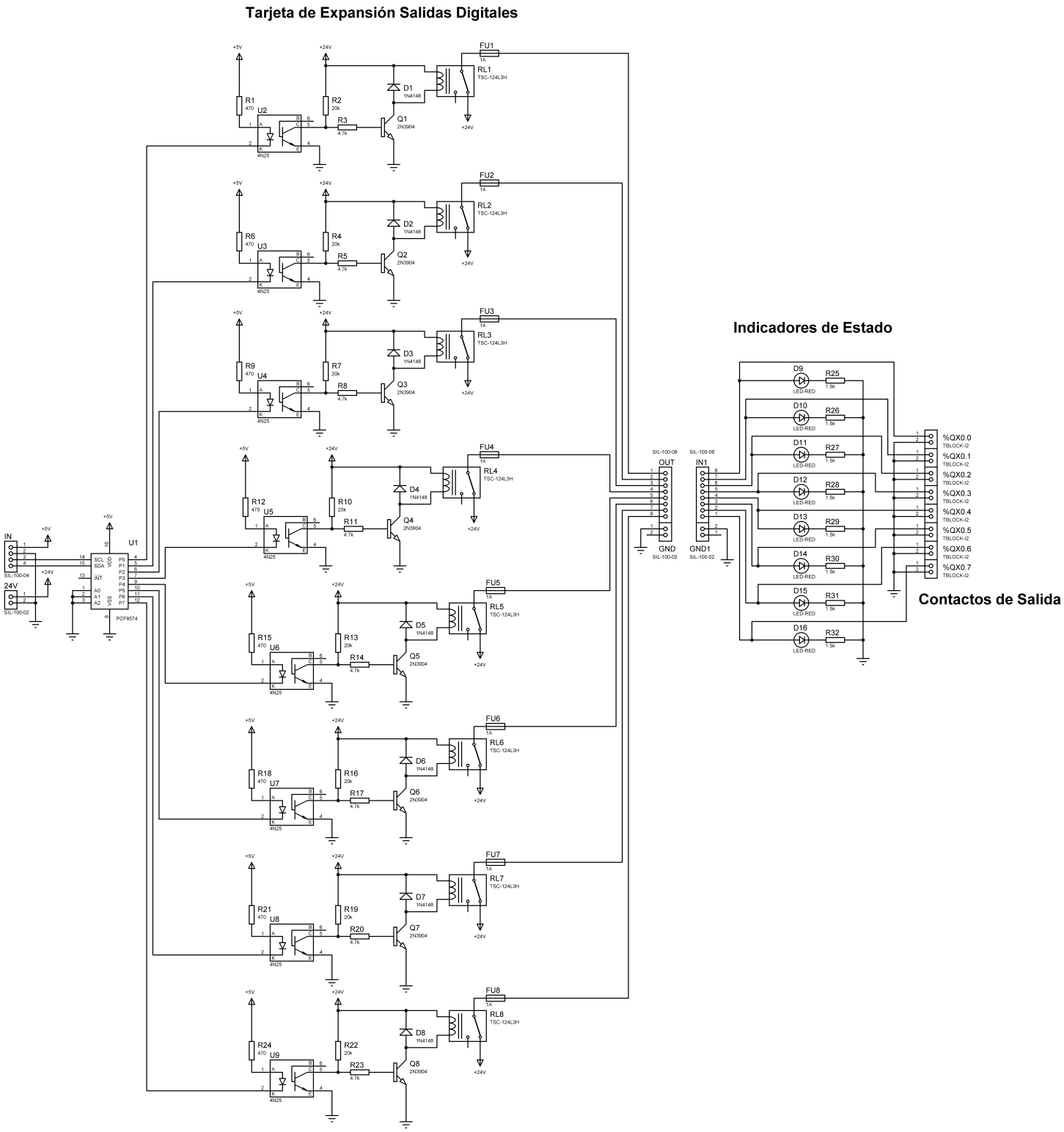
ANEXOS

A. ESQUEMÁTICOS ELECTRÓNICOS

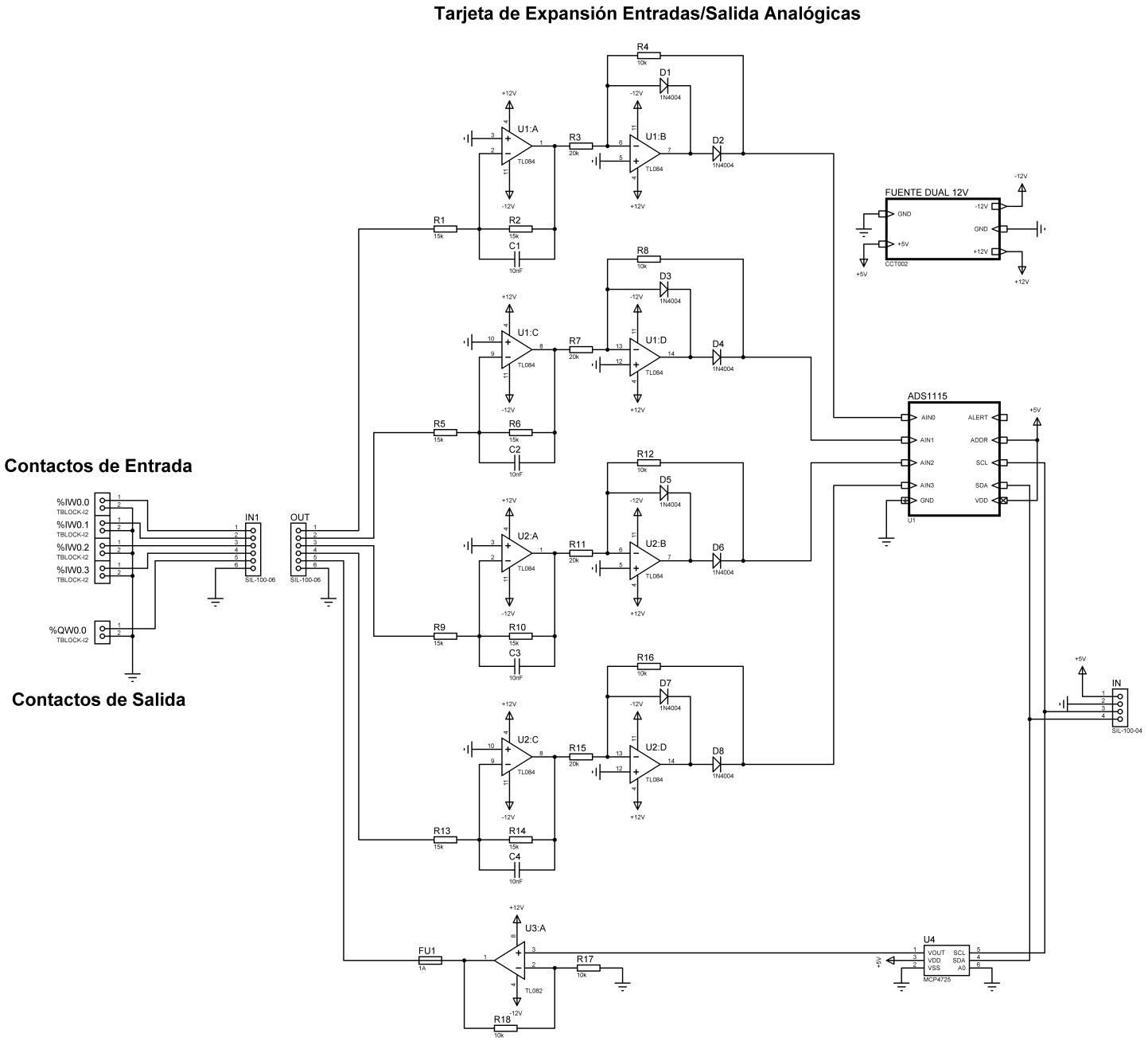
A-1. Módulo de Entradas Digitales



A-2 Módulo de Salidas Digitales



A-3 Módulo de Entradas/Salidas Analógicas



B. ELEMENTOS DE SOFTWARE

B-1. Capa de Hardware OpenPi Controller

```
/* Capa de hardware OpenPiControler para OpenPLC */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <wiringPi.h>
#include <wiringPiI2C.h>
#include <pcf8574.h>
#include <ads1115.h>

#define DOUT_PINBASE                    100
#define DIN_PINBASE                     90
#define ADC_PINBASE                     65
#define DAC_PINBASE                     75
#define MAX_INPUT                       8
#define MAX_OUTPUT                      8

int adcBuffer[4];
int data[2]; // DAC

void *readAdcThread(void *args)
{
    while(1)
    {
        for(int i=0; i<4; i++)
        {
            pthread_mutex_lock(&adcBufferLock);
            adcBuffer[i] = analogRead(ADC_PINBASE + i);
            pthread_mutex_unlock(&adcBufferLock);
        }
    }
}
//-------------------------------------------------------------------------
// ADS1115 Reading
//-------------------------------------------------------------------------
int ads_adcRead(int chan)
{
        int returnValue;
        if (chan < 4)
        {
            pthread_mutex_lock(&adcBufferLock);
            returnValue = adcBuffer[chan];
            pthread_mutex_unlock(&adcBufferLock);
            return returnValue;
        }
        return 0;
}
//-------------------------------------------------------------------------
// This function is called to initialize MCP4725
//-------------------------------------------------------------------------
void mcp4725Setup(int addr)
{
        dac_fd = wiringPiI2CSetup(addr);
}
//-------------------------------------------------------------------------
// MCP4725 Writing
//-------------------------------------------------------------------------
void mcp_dacWrite(int value)
{
        data[0] = (value >> 8) & 0xFF; // [0 0 0 0 D11 D10 D9 D8] (first bits are modes for our use 0 is fine)
        data[1] = value; // [D7 D6 D5 D4 D3 D2 D1 D0]
        wiringPiI2CWrite(dac_fd, 0x40);
        wiringPiI2CWriteReg8(dac_fd, data[0], data[1]);
}
//-------------------------------------------------------------------------
// This function is called by the main OpenPLC routine when it is initializing.
// Hardware initialization procedures should be here.
//-------------------------------------------------------------------------
void initializeHardware()
{
        pcf8574Setup(DOUT_PINBASE, 0x20); // Digital Output Expansion
        pcf8574Setup(DIN_PINBASE, 0x21); // Digital Input Expansion
        ads1115Setup(ADC_PINBASE, 0x49); // ADC Expansion
        mcp4725Setup(0x60); // DAC Expansion

        for (int i = 0; i < MAX_OUTPUT; i++)
        {
            pinMode(DOUT_PINBASE + i, OUTPUT);
        }

        for (int i = 0; i < MAX_INPUT; i++)
        {
            pinMode(DIN_PINBASE + i, INPUT);
        }

        pthread_t ADCthread;
        pthread_create(&ADCthread, NULL, readAdcThread, NULL);
}
//-------------------------------------------------------------------------
// This function is called by the OpenPLC in a loop. Here the internal buffers
// must be updated to reflect the actual I/O state. The mutex buffer_lock
// must be used to protect access to the buffers on a threaded environment.
//-------------------------------------------------------------------------
void updateBuffers()
{
        pthread_mutex_lock(&bufferLock); //lock mutex

        for (int i = 0; i < MAX_INPUT; i++)
        {
                if (bool_input[i/8][i%8] != NULL) *bool_input[i/8][i%8] = !digitalRead(DIN_PINBASE + i);
        }

        for (int i = 0; i < MAX_OUTPUT; i++)
        {
                if (bool_output[i/8][i%8] != NULL) digitalWrite(DOUT_PINBASE + i, *bool_output[i/8][i%8]);
        }

        for (int i = 0; i < 4; i++)
        {
                if (int_input[i] != NULL) *int_input[i] = ads_adcRead(i);
        }

        // Analog OUT.
        if (int_output[0] != NULL) mcp_dacWrite((*int_output[0] / 16.0));

        pthread_mutex_unlock(&bufferLock); //unlock mutex
}
```

B-2. Clase ModbusStream

```python
# ModbusMonitor.py - Definición de clase para un Stream que supervisa cambios de estado
#                    en el PLC

import threading
from pymodbus3.client.sync import ModbusTcpClient


class ModbusStream(threading.Thread):

    def __init__(self, host, port, database, target, callback):

        threading.Thread.__init__(self)
        self.client = ModbusTcpClient(host, port)
        self.callback = callback
        self.db = database
        self.target = target

    def run(self):
        """Detección de cambios en estados del PLC"""

        try:
            if self.target == 'coils':
                old = self.client.read_coils(address=0, count=8).bits
            elif self.target == 'holding_registers':
                old = self.client.read_holding_registers(address=0, count=1).registers
            elif self.target == 'digital_inputs':
                old = self.client.read_discrete_inputs(address=0, count=8).bits
            elif self.target == 'input_registers':
                old = self.client.read_input_registers(address=0, count=4).registers
            else:
                return

            while True:

                if self.target == 'coils':
                    new = self.client.read_coils(address=0, count=8).bits
                elif self.target == 'holding_registers':
                    new = self.client.read_holding_registers(address=0, count=1).registers
                elif self.target == 'digital_inputs':
                    new = self.client.read_discrete_inputs(address=0, count=8).bits
                elif self.target == 'input_registers':
                    new = self.client.read_input_registers(address=0, count=4).registers

                if new != old:
                    self.callback(new, self.db, self.target)

                old = new

        except Exception as e:
            print(e)
```

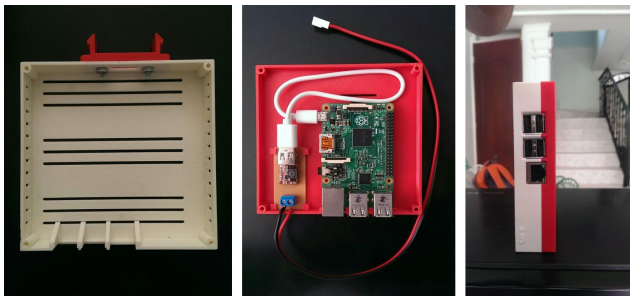
B-3. Enlace OpenPi Controller con Firebase

```python
# RemotePiCloud.py - Script principal. Inicializa Stream de datos en el PLC y Firebase.

import pyrebase
import time
from pymodbus3.client.sync import ModbusTcpClient
from ModbusMonitor import ModbusStream


def sync_firebase():
    """Sincroniza Firebase con el estado actual del PLC al iniciar el script"""
    coils = client.read_coils(address=0, count=8).bits
    inputs = client.read_discrete_inputs(address=0, count=8).bits
    hregisters = client.read_holding_registers(address=0, count=1).registers[:1]
    iregisters = client.read_input_registers(address=0, count=4).registers[:4]

    modbus_stream_handler(status=coils, database=db, target='coils')
    modbus_stream_handler(status=inputs, database=db, target='digital_inputs')
    modbus_stream_handler(status=hregisters, database=db, target='holding_registers')
    modbus_stream_handler(status=iregisters, database=db, target='input_registers')
    print('Sync Done!\n')


def firebase_coils_handler(message):
    """Envío de datos desde Firebase hacia el PLC"""
    try:
        status = db.child('/Coils').get().val()
        print('{0} - Firebase Status: {1}\n'.format(time.strftime('%H:%M:%S'), dict(status)))
        [print('{}: {}'.format(key, status[key])) for key in status]
        print('\n')
        data = [status['QX0{0}'.format(i)] for i in range(8)]
        client.write_coils(address=0, values=data)
    except Exception as e:
        print('{0} - {1}\n'.format(time.strftime('%H:%M:%S'), e))


def firebase_hregisters_handler(message):
    """Envío de datos desde Firebase hacia el PLC"""
    try:
        status = db.child('/HoldingRegisters').get().val()
        print('{0} - Firebase Status: {1}\n'.format(time.strftime('%H:%M:%S'), dict(status)))
        [print('{}: {}'.format(key, status[key])) for key in status]
        print('\n')
        data = [status['QW0']]
        client.write_registers(address=0, values=data)
    except Exception as e:
        print('{0} - {1}\n'.format(time.strftime('%H:%M:%S'), e))


def modbus_stream_handler(status, database, target):
    """Envío de datos desde PLC hacia la nube"""
    if target == 'coils':
        data = {
            'QX00': status[0],
            'QX01': status[1],
            'QX02': status[2],
            'QX03': status[3],
            'QX04': status[4],
            'QX05': status[5],
            'QX06': status[6],
            'QX07': status[7],
        }
        database.child('/Coils').set(data)

    elif target == 'holding_registers':
        data = { 'QW0': status[0] }
        database.child('/HoldingRegisters').set(data)

    elif target == 'digital_inputs':
        data = {
            'IX00': status[0],
            'IX01': status[1],
            'IX02': status[2],
            'IX03': status[3],
            'IX04': status[4],
            'IX05': status[5],
            'IX06': status[6],
            'IX07': status[7],
        }
        database.child('/DigitalInputs').set(data)

    elif target == 'input_registers':
        data = {
            'IW00': status[0],
            'IW01': status[1],
            'IW02': status[2],
            'IW03': status[3],
        }
        database.child('/InputRegisters').set(data)
    else:
        print('Invalid Target!')
        return

    print('{0} - Sent to Firebase: {1}\n'.format(time.strftime('%H:%M:%S'), str(data)))
    [print('{}: {}'.format(key, data[key])) for key in data]
    print('\n')


client = ModbusTcpClient('localhost')

config = {
        "apiKey": "AIzaSyD4QSutZ1KONhtY4j1uUG7zAlbcbWdR2QA ",
        "authDomain": "picontroller-d327b.firebaseapp.com",
        "databaseURL": "https://picontroller-d327b.firebaseio.com/",
        "storageBucket": "picontroller-d327b.appspot.com"
    }

firebase = pyrebase.initialize_app(config)
auth = firebase.auth()
db = firebase.database()

sync_firebase()
fire_coil_stream = db.child("/Coils").stream(firebase_coils_handler)
fire_hregister_stream = db.child('/HoldingRegisters').stream(firebase_hregisters_handler)


coils_stream = ModbusStream(host='localhost', port=502, database=db,
                            target='coils', callback=modbus_stream_handler)

hregisters_stream = ModbusStream(host='localhost', port=502, database=db,
                                 target='holding_registers', callback=modbus_stream_handler)

dinputs_stream = ModbusStream(host='localhost', port=502, database=db,
                                 target='digital_inputs', callback=modbus_stream_handler)

iregisters_stream = ModbusStream(host='localhost', port=502, database=db,
                                 target='input_registers', callback=modbus_stream_handler)

coils_stream.start()
hregisters_stream.start()
dinputs_stream.start()
iregisters_stream.start()
```

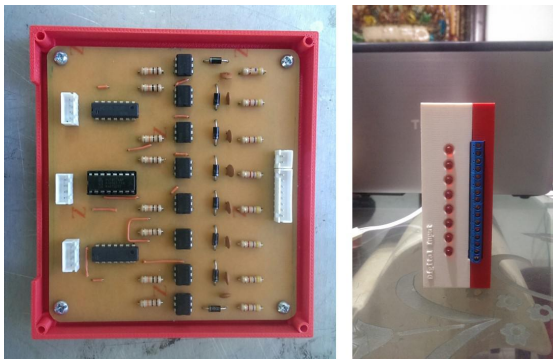
C. ENSAMBLE DEL PROTOTIPO

Fuente de Alimentación

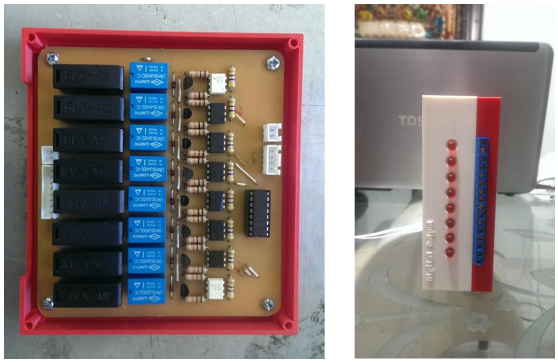
Unidad Central de Proceso



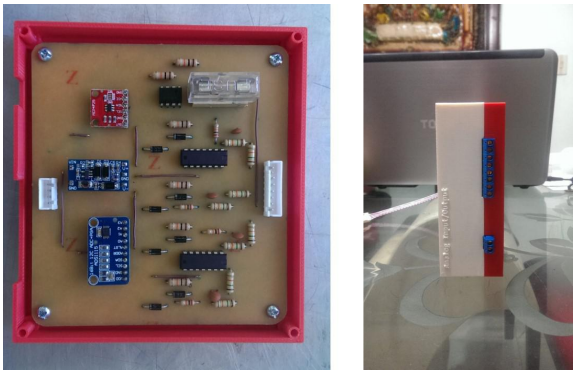
Módulo Entradas Digitales



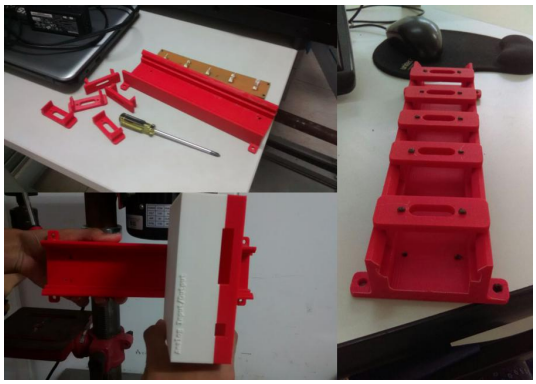
Módulo Salidas Digitales

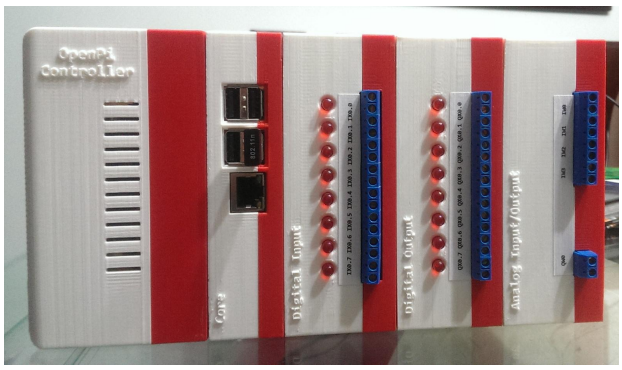


Módulo E/S Analógicas



Riel de Soporte e Interconexión





OpenPi Controller
Estructura Final