INSTITUTO TECNOLÓGICO SUPERIOR DE ATLIXCO

Organismo Público Descentralizado del Gobierno del Estado de Puebla

NOMBRE DEL TRABAJO

Desarrollo de Interfaz de programación para PLC

OPCIÓN

Tesis

QUE PARA OBTENER EL TÍTULO DE:

Ingeniero Mecatrónico

P R E S E N T A:
José Carlos Mundo Infante

ASESOR: **Ing. Raúl Eusebio Grande**

ATLIXCO, PUE. AGOSTO DE 2022

Contenido general

Figuras. *4*

Tablas. *6*

Introducción. *7*

Planteamiento del problema. *8*

Justificación. *8*

Objetivos. *9*

Alcances y limitaciones. *9*

Capítulo 1. Estado del arte. *10*

1.1. Open PLC. **10**

1.2. PLC Industrial Arduino. **10**

1.3. PLC Controllino. **11**

Capítulo 2. Marco teórico. *12*

2.1. Microcontrolador STM32F407 ..... **12**

2.2. Comunicación Serial **12**

Módulo USART. 13

2.3. MicroPython **14**

¿Qué es MicroPython? 14

PyBoard 14

GPIO 16

UART 17

2.4. Python **18**

Tkinter, Interfaz de Python para Tcl/Tk. 18

Comunicación serial en Python 20

2.5. Ophyra **21**

Puertos de propósito general y su configuración. 22

Bus de datos USB-OTG. 22

Bus de datos del puerto COM. 24

Pulsadores y Led RGB. 24

2.6. Instrucciones de un Controlador Lógico Programable **25**

Funciones lógicas. 25

Temporización. 26

Contaje. 29

Capítulo 3. Desarrollo. *31*

3.1. Arquitectura del sistema. 31
3.2. Lectura de entradas y salidas. 31
3.3. Implementación de condicional if. 33
Lectura en la interfaz. 33
Lectura en MicroPython. 35
3.4. Implementación del temporizador. 37
Lectura en la interfaz. 37
Lectura en Micropython. 38
3.5. Implementación del contador. 39
Lectura en la interfaz. 39
Lectura en MicroPython. 40
3.6. Implementación de lectura analógica y recepción de datos. 41
Lectura en la interfaz. 41
Lectura en MicroPython. 42
Activación y preparación para recepción de información en la interfaz. 43
Capítulo 4. Resultados. 44
4.1. Uso y habilitación de la comunicación serial. 44
4.2. Declaración de entradas y salidas. 45
4.3. Uso de los elementos de programación. 47
Función lógica. 47
Temporizador. 48
Contador 49
Convertidor analógico-Digital. 50
4.4. Uso de la interfaz en diferentes computadoras con archivo exe. 52
Conclusión. 53
Mejoras a futuro. 55
Referencias 56
Anexos 57
4.5. Comunicación serial entre Python y STM32 Micropython. 57
4.6. Declaración de algunos Widgets en interfaz TKinter 59
Botón (Button) en Tcl/Tk (tkinter) 59
Caja de texto (Entry) en Tcl/Tk (tkinter) 60
Lista desplegable (Combobox) en Tcl/Tk (tkinter) 61
4.7. Convertir un archivo py a exe. 63

Figuras.

FIGURA 1. LOGO DEL PROYECTO OPEN PLC. (ALVES, 2022) 10
FIGURA 2. PLC INDUSTRIAL ARDUINO PARA AUTOMATIZACIÓN. (INDUSTRIAL SHIELDS, 2022) 11
FIGURA 3. PLC CONTROLLINO CON SOFTWARE DE CÓDIGO ABIERTO. (CONTROLADORES, 2022) 11
FIGURA 4. COMUNICACIÓN ASÍNCRONA. (MAOCHO, 2016) 14
FIGURA 5. LA PYBOARD ORIGINAL. (TOLLERVEY, 2017) 15
FIGURA 6. FUNCIONAMIENTO DE LOS PINES EN EL PYBOARD. (IOTTRENDS.TECH, 2020) 18
FIGURA 7. DIAGRAMA A BLOQUES DEL SISTEMA OPHYRA. (INTESC ELECTRONICS & EMBEDDED, 2017) 22
FIGURA 8. DIAGRAMA A BLOQUES DEL BUS DE DATOS USB-OTG. (INTESC ELECTRONICS & EMBEDDED, 2017) 23
FIGURA 9. DIAGRAMA A BLOQUES DEL PUERTO COM. (INTESC ELECTRONICS & EMBEDDED, 2017) 24
FIGURA 10. SÍMBOLO LÓGICO DE UN TEMPORIZADOR. (PÉREZ, 2009) 26
FIGURA 11. CRONOGRAMA DE UN TEMPORIZADOR DE IMPULSO. (PÉREZ, 2009) 27
FIGURA 12. CRONOGRAMA DE UN TEMPORIZADOR DE IMPULSO PROLONGADO. (PÉREZ, 2009) 28
FIGURA 13. CRONOGRAMA DE UN TEMPORIZADOR CON RETARDO A LA CONEXIÓN. (PÉREZ, 2009) 28
FIGURA 14. CRONOGRAMA DE UN TEMPORIZADOR CON RETARDO A LA CONEXIÓN CON MEMORIA. (PÉREZ, 2009) 29
FIGURA 15. CRONOGRAMA DE UN TEMPORIZADOR CON RETARDO A LA DESCONEXIÓN. (PÉREZ, 2009) 29
FIGURA 16. SÍMBOLO LÓGICO DE UN TEMPORIZADOR. (PÉREZ, 2009) 30
FIGURA 17. ESQUEMA GENERAL DE CONEXIONES ENTRE UNA PC Y UN STM32. 31
FIGURA 18. DIAGRAMA DE FLUJO PARA LA LECTURA DE LAS ENTRADAS EN LA INTERFAZ. 32
FIGURA 19. DIAGRAMA DE FLUJO PARA EL ENVÍO DE LOS PARÁMETROS DE LA FUNCIÓN LÓGICA. 34
FIGURA 20. DIAGRAMA DE FLUJO PARA LA DECODIFICACIÓN DE LOS PARÁMETROS DE LA FUNCIÓN LÓGICA. 36
FIGURA 21. DIAGRAMA DE FLUJO PARA LA ACTIVACIÓN Y DESACTIVACIÓN DE LA SALIDA. 36
FIGURA 22. DIAGRAMA DE FLUJO PARA EL ENVÍO DE LOS PARÁMETROS DEL TEMPORIZADOR. 37
FIGURA 23. DIAGRAMA DE FLUJO DE LA DECODIFICACIÓN DE LOS PARÁMETROS PARA EL TEMPORIZADOR. 39
FIGURA 24. DIAGRAMA DE FLUJO PARA EL ENVÍO DE LOS PARÁMETROS DEL CONTADOR. 40
FIGURA 25. DIAGRAMA DE FLUJO DE LA DECODIFICACIÓN DE LOS PARÁMETROS DEL CONTADOR. 41
FIGURA 26. DIAGRAMA DE FLUJO PARA EL ENVÍO DE LOS PARÁMETROS DEL ADC. 42
FIGURA 27. DIAGRAMA DE FLUJO PARA EL ENVÍO DE PARÁMETROS DEL PIN ANALÓGICO. 43
FIGURA 28. DIAGRAMA DE FLUJO PARA LA ACTIVACIÓN DE LA LECTURA DE DATOS. 43
FIGURA 29. DIAGRAMA DE FLUJO PARA LA DESACTIVACIÓN DE LA LECTURA DE DATOS. 44
FIGURA 30. UBICACIÓN Y SELECCIÓN DEL BOTÓN PARA BUSCAR PUERTOS COM. 44
FIGURA 31. SELECCIÓN DEL PUERTO COM (ARRIBA), SELECCIÓN DE LA VELOCIDAD EN BAUDIOS (ABAJO). 45
FIGURA 32. MENSAJE AL ESTAR ACTIVADA LA COMUNICACIÓN SERIAL. 45

FIGURA 33. SELECCIÓN DEL NÚMERO DE ENTRADAS PARA LA FUNCIÓN LÓGICA. 46
FIGURA 34. ESCRITURA EN LOS CUADROS DE TEXTO CON EL NOMBRE DE LOS PINES A UTILIZAR. 46
FIGURA 35. INTRODUCCIÓN DE LOS PINES PARA EL TEMPORIZADOR Y EL CONTADOR. 46
FIGURA 36. VISUALIZACIÓN DE LAS ENTRADAS Y SALIDAS DECLARADAS PARA CADA UNO DE LOS ELEMENTOS DE PROGRAMACIÓN. 47
FIGURA 37. NÚMERO DE VARIABLES PARA LA FUNCIÓN LÓGICA. 47
FIGURA 38. FORMATO DE LA FUNCIÓN LÓGICA AL USAR EL BOTÓN “GENERAR REGLA”. 47
FIGURA 39. FUNCIÓN CON AND Y ENTRADAS NA (IZQUIERDA), FUNCIÓN CON OR Y ENTRADAS NC (DERECHA). 48
FIGURA 40. PIN DE RESET DESACTIVADO (SIN MARCAR CASILLA), PIN DE RESET ACTIVADO (CASILLA MARCADA). 48
FIGURA 41. SELECCIÓN DEL TIPO DE TEMPORIZADOR A EMPLEAR. 48
FIGURA 42. SELECCIÓN DE LA ENTRADA DEL TEMPORIZADOR. 49
FIGURA 43. COLOCACIÓN DEL TIEMPO EN SEGUNDOS Y SELECCIÓN DEL PIN DE SALIDA. 49
FIGURA 44. SELECCIÓN DEL TIPO DE CONTADOR. 49
FIGURA 45. SELECCIÓN DE LOS TRES PINES DE ENTRADA PARA EL INCREMENTO, DECREMENTO Y RESET. 50
FIGURA 46. CARGA EN NÚMERO ENTERO Y EL PIN DE SALIDA. 50
FIGURA 47. SELECCIÓN DEL PIN ANALÓGICO. 50
FIGURA 48. SELECCIÓN DE LA CONDICIÓN <= Ó >=. 51
FIGURA 49. ESCRITURA DEL VALOR DE REFERENCIA DEL ADC, ACTIVAR SALIDA CUANDO EL VALOR REAL DEL ADC ES MAYOR AL DE REFERENCIA (IZQUIERDA) Y DESACTIVAR SALIDA CUANDO EL VALOR REAL DEL ADC ES MENOR AL DE REFERENCIA (DERECHA). 51
FIGURA 50. BOTÓN PARA CARGAR EL PROGRAMA AL MICROCONTROLADOR AL TERMINAR LA PROGRAMACIÓN. 51
FIGURA 51. VISUALIZACIÓN DEL CÓDIGO AL CARGAR LA PROGRAMACIÓN EN EL MICROCONTROLADOR. 52
FIGURA 52. ARCHIVO EXE EN UNA COMPUTADORA DE 32 BITS MARCA ACER® SISTEMA OPERATIVO WINDOWS. 52
FIGURA 53. ARCHIVO EXE EN UNA COMPUTADORA DE 64 BITS MARCA HP® SISTEMA OPERATIVO WINDOWS. 53
FIGURA 54. IDENTIFICACIÓN DE LOS PUERTOS COM DEL MICROCONTROLADOR. 57
FIGURA 55. CREACIÓN DE LA CARPETA INTERFAZPLC. 63
FIGURA 56. COPIA DEL ARCHIVO PY DEL PROYECTO A LA CARPETA RECIÉN CREADA. 64
FIGURA 57. ARCHIVO DE PYTHON JUNTO A SUS COMPONENTES O IMÁGENES. 65
FIGURA 58. CMD DESDE PYCHARM. 65
FIGURA 59. COMANDO INTRODUCIDO PARTA COMENZAR CON EL PROCESO DE CONVERSIÓN. 66
FIGURA 60. NUEVOS ARCHIVOS GENERADOS POR EL COMANDO PYINSTALLER. 67
FIGURA 61. ARCHIVO EJECUTABLE JUNTO A LAS IMÁGENES. 67
FIGURA 62. INTERFAZ DEL ARCHIVO EJECUTABLE. 68

Tablas.

TABLA 1. TABLA DE LOS PARÁMETROS QUE SE PUEDEN ESPECIFICAR EN LA CLASE SERIAL DE PYTHON. 21
TABLA 2. DISPOSICIÓN DE LOS PINES PARA EL BUS DE DATOS USB-OTG. (INTESC ELECTRONICS & EMBEDDED, 2017) 23
TABLA 3. DISPOSICIÓN DE LOS PINES DEL PUERTO COM. (INTESC ELECTRONICS & EMBEDDED, 2017) 24
TABLA 4. DISPOSICIÓN DE LOS PINES DE LOS PULSADORES Y EL LED RGB. (INTESC ELECTRONICS & EMBEDDED, 2017) 25
TABLA 5. EJEMPLOS DE ALGUNAS OPERACIONES CONTENIDAS EN LA CLASE ENTRY. 61
TABLA 6. DIFERENCIA ENTRE LA IMAGEN CUANDO ESTA EN EL MISMO DIRECTORIO Y CUANDO ESTÁ EN OTRO DIRECTORIO. 64

Introducción.

El controlador lógico programable, autómata programable o simplemente PLC, tiene como tarea la interconexión de las señales de entrada que se encuentran regidas por un determinado programa y de acuerdo con la combinación de dichas entradas puede dar como resultado que una salida se encuentre activada (TRUE) o desactivada (FALSE).

Según la IEC 1131 define el término “Controlador Lógico Programable” como: "Un sistema electrónico de funcionamiento digital, diseñado para ser utilizado en un entorno industrial, que utiliza una memoria programable para el almacenamiento interno de instrucciones orientadas al usuario, para la realización de funciones específicas tales como enlaces lógicos, secuenciación, temporización, recuento y cálculo, para controlar, a través de entradas y salidas digitales o analógicas, diversos tipos de máquinas o procesos”. Por lo tanto, un PLC es un ordenador adaptado a ciertas tareas de control. (FESTO DIDACTIC, 2002)

Pero antes del PLC estaba el microcontrolador que es un circuito lógico programable que contiene todos los elementos necesarios para controlar el funcionamiento de una tarea específica. Un sistema con microcontrolador debe disponer de una memoria donde se almacena el programa que gobierna el funcionamiento de este que, una vez programado y configurado, solo sirve para realizar la tarea asignada.

Ambos parecen ser similares pero su mayor diferencia radica en la programación de cada uno de ellos; el PLC tiene mayormente métodos de programación gráfica y el microcontrolador requiere de una más sofisticada mediante un programa. Actualmente ambos siguen siendo muy utilizados en la industria, pero el que más aplicaciones tiene es el PLC porque facilita la programación de un proceso automatizado solo que a diferencia del microcontrolador es de 20 a 50 veces más costoso y su uso se encuentra limitado por un software propio con licencia, el microcontrolador comúnmente su programación es libre por lo que hoy en día se dio paso a un nuevo método mediante el uso de un software libre y un microcontrolador realizar una interconexión que realice la programación de microcontrolador como si de un PLC se tratase.

Planteamiento del problema.

La automatización industrial facilita y mejora los procesos de todo proceso automático, todo eso es gracias al muy conocido PLC (Controlador Lógico Programable), que no es más que un microcontrolador caracterizado por su maneras versátiles de programación ya que son entendibles casi para cualquiera, pero a pesar de que la programación de un controlador lógico programable puede ser de sencillas y distintas maneras, con solo el uso de conceptos básicos de lógica digital, el uso de los PLC's es muy costoso debido a que estos dispositivos funcionan a base del uso de licencias para un software específico de programación más el equipo que se va a programar; por otro lado, el uso de un microcontrolador de tipo PIC o ARM es de un bajo costo que están a disponibilidad de la mayoría de los usuarios, pero su programación enfocada a la automatización requiere de experiencia y conocimiento avanzado en la rama de la lógica digital lo que llega a dificultar su aplicación.

Justificación.

Un PLC al ser de un alto costo y que para usarlo se requiere de una licencia de un software con un mayor precio, esto deja claro que no muchos puedan hacer uso de este dispositivo pero si se tiene la oportunidad de usarlo su programación esta es muy intuitiva y fácil de emplear; por otro lado el microcontrolador es de bajo costo y los software que permiten su programación se encuentran gratis o de fácil adquisición pero usarlo es más complicado puesto que requiere de más instrucciones estudios más avanzados.

Por lo que no deja muchas opciones el querer programar un controlador, ya sea que uno esté dispuesto a gastar lo de un mes o emplear bastante tiempo en aprender a programar, así que con la tercera opción de unir ambos beneficios de ambos dispositivos sería un camino muy fiable; la facilidad de programación del PLC y el costo de un microcontrolador.

Objetivos.

- **Desarrollar una interfaz de usuario que permita la programación de manera gráfica tanto lógica como de los diferentes periféricos del microcontrolador STM32F40**.
- Investigar sobre la programación en lenguaje Python y MicroPython.
- Implementar la comunicación entre una PC y un microcontrolador a través de Python.
- Desarrollar la lógica para instrucciones condicionales.
- Desarrollar la lógica para temporizadores y contadores.
- Implementar las instrucciones condicionales, temporizadores y contadores con una interfaz gráfica en Python.
- Realizar pruebas de funcionamiento.
- Documentar por escrito el trabajo realizado.

Alcances y limitaciones.

Alcances:

- La interfaz va dirigida a cualquier operador que cuente con conocimientos básicos de programación.
- Control de todos los pines GPIO mediante la interfaz.
- Uso de los tres elementos básicos de programación avanzada (condicionales, temporización y contaje).
- La interfaz llega hasta la implementación física de un proceso de automatización básico.

Limitaciones:

- La programación está centrada a un solo tipo de microcontrolador.
- Aplicaciones limitadas y básicas debido al número de instrucciones.
- No se pueden usar todos los pines del microcontrolador que no sean GPIO.

Capítulo 1. Estado del arte.

1.1. Open PLC.

Es el primer PLC estandarizado de código abierto totalmente funcional, tanto en software como en hardware. Así que se trata un controlador lógico programable de código abierto que se basa en un software fácil de usar. OpenPLC fue creado de acuerdo con el estándar IEC 61131-3, que define la arquitectura de software básica y los lenguajes de programación para PLC.

Figura 1. *Logo del proyecto Open PLC. (Alves, 2022)*

OpenPLC se utiliza principalmente en la industria y la automatización del hogar, Internet de las cosas y la investigación SCADA.

Consta de tres partes: Runtime, Editor y HMI Builder. El tiempo de ejecución debe estar instalado en su dispositivo y es responsable de ejecutar su programa PLC. El Editor es el software que se ejecuta en su computadora y se utiliza para crear sus programas PLC. Finalmente, ScadaBR es el HMI Builder. ScadaBR se comunica con OpenPLC Runtime a través de Modbus/TCP. (Alves, 2022)

1.2. PLC Industrial Arduino.

Este autómata basado en Arduino (Open Source Hardware) está especialmente diseñado para su uso en un entorno profesional. Este autómata dispone de hasta 20 Entradas/Salidas, también dispone de diferentes sistemas de comunicación lo que le ofrece una gran flexibilidad y control. El PLC 20IOs ofrece la posibilidad de expandirse con 127 módulos mediante el sistema I2C, lo que significa que puede gobernar hasta 7100 E/S en modo maestro esclavo, además de módulos adicionales de sensores, etc.

Este PLC industrial se puede programar utilizando la plataforma Arduino IDE y otros lenguajes o interfaces de programación, ya que es de código abierto. (Industrial Shields, 2022)

Figura 2. *PLC industrial Arduino para automatización. (Industrial Shields, 2022)*

1.3. PLC Controllino.

Es una placa industrializada compatible con Arduino que tiene el objetivo de que el mundo de la industria y la automatización sea más abierto, independiente, flexible, innovador, seguro y libre de licencias.

Se puede programar con Arduino IDE, así como con muchas otras soluciones de software como Visuino, OpenPLC, logi.CAD, LabVIEW, GNU Octave, Matlab, Atmel Studio, etc. (Controladores, 2022)

Figura 3. *PLC Controllino con software de código abierto. (Controladores, 2022)*

Capítulo 2. Marco teórico.

2.1. Microcontrolador STM32F407

La familia STM32F405xx y STM32F407xx se basa en el núcleo RISC de 32 bits Arm Cortex-M4® de alto rendimiento que funciona a una frecuencia de hasta 168 MHz.

El núcleo Cortex-M4 presenta una unidad de punto flotante (FPU) de precisión simple que admite todas las instrucciones de procesamiento de datos y tipos de datos de precisión simple de Arm. También implementa un conjunto completo de instrucciones DSP y una unidad de protección de memoria (MPU) que mejora la seguridad de la aplicación.

La familia STM32F405xx y STM32F407xx incorpora memorias integradas de alta velocidad, hasta 4 Kbytes de SRAM de respaldo y una amplia gama de E/S mejoradas y periféricos conectados a dos APB buses, tres buses AHB y una matriz de bus multi-AHB de 32 bits.

Todos los dispositivos ofrecen tres ADC de 12 bits, dos DAC, un RTC de bajo consumo, doce temporizadores de 16 bits de uso general, incluidos dos temporizadores PWM para control de motores, dos temporizadores de 32 bits de uso general, un verdadero generador de números aleatorios (RNG). También cuentan con interfaces de comunicación estándar y avanzadas. (STMicroelectronics, 2022)

2.2. Comunicación Serial

Los PIC en general utilizan dos métodos de transmisión en serie:

- El puerto serie síncrono o SSP.

  Este se suele usar en la comunicación con otros microcontroladores o con periféricos que consta de dos interfaces de trabajo:

  - Interfaz serie de periféricos (SPI), para la comunicación entre microcontroladores en modo maestro-esclavo; *Full-duplex*.
  - Interfaz Inter-Circuitos (I2C), cuenta con una gran capacidad para comunicar microcontroladores y periféricos; *Half-Duplex*.

- La interfaz de comunicación serie (SCI) o el receptor transmisor serie síncrono-asíncrono universal (USART).
  Esta configuración permite la comunicación con un ordenador trabajando en modo *full-duplex* asíncrono o con periféricos trabajando en modo *half-duplex*.

Otros tipos de comunicación que son soportados por los PIC son: *1-Wire bus*, *LIN*, *USB*, el *CAN* y *Ethernet*.

Módulo USART.

Esta operación se divide en dos categorías: síncrona o asíncrona. La transmisión síncrona utiliza una señal de reloj y una línea de datos, por otra parte, en la transmisión asíncrona no se envía una señal de reloj, por lo que el emisor y el receptor deben tener relojes con la misma frecuencia y fase. Si la distancia entre emisor y receptor es pequeña se utiliza la comunicación síncrona, mientras que para distancias mayores se utiliza la comunicación asíncrona.

El USART puede transmitir o recibir datos serie ya sea en un conjunto de 8 o 9 bits por transmisión y detectar errores en el intercambio de datos. También puede generar interrupciones cuando se detecta una recepción de datos o cuando el proceso de comunicación se complete.

De manera general, la comunicación serie permite enviar los datos bit a bit a través de una línea común en periodos de tiempo fijos, lo que da lugar a la velocidad de comunicación o número de bits enviados por segundo, denominados baudios.

El **modo asíncrono** emplea los relojes de emisor y receptor que, como se mencionaba anteriormente, deben estar sincronizados en frecuencia y fase. La frecuencia del reloj es configurada antes de la transmisión y la sincronización se realiza durante la comunicación. Cada conjunto de datos tiene un tamaño fijo y poseen un bit de arranque (*start*) y un bit de parada (*stop*) lo que permite lograr la sincronización. Como se trata de una transmisión en modo *full-duplex* se utilizan dos líneas, la de transmisión TX y la de recepción RX. (Breijo, 2008)

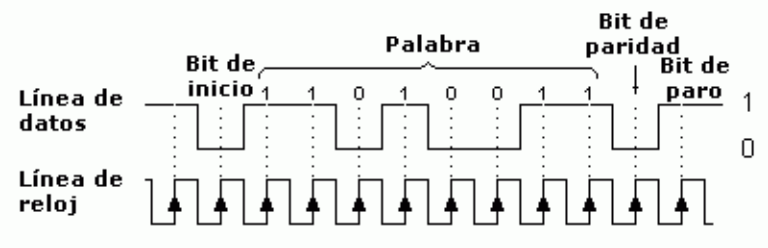

Figura 4. *Comunicación asíncrona. (Maocho, 2016)*

2.3. MicroPython

¿Qué es MicroPython?

Se trata de una reimplementación sencilla y eficiente del lenguaje de programación Python aplicado a una biblioteca estándar optimizada para su uso en microcontroladores y sistemas embebidos. A diferencia de Python, MicroPython trabaja en condiciones limitadas.

MicroPython se ejecuta en “bare-metal” directamente en el hardware, es decir, no cuenta con un sistema operativo subyacente, por lo que Micropython tiene el control total y directo del hardware lo que hace a Micropython ser el sistema operativo.

Regularmente no viene con la biblioteca estándar completa de forma predeterminada. Depende enteramente del dispositivo el cual viene con un subconjunto de la biblioteca estándar. Todas las versiones de MicroPython vienen con módulos para interactuar el hardware como los pines GPIO (entrada/salida de propósito general) periféricos y componentes. (Tollervey, 2017)

PyBoard

El PyBoard es el primer dispositivo desarrollado y construido para Micropython la cual se conecta vía micro USB a una PC, esta conexión provee dos maneras de interactuar con el dispositivo: como unidad flash USB y como Python REPL (bucle Lectura-Evaluación-Impresión) basado en serie.

PyBoard contiene un pequeño grupo de archivos de sistema como parte de su memoria flash, además tiene una ranura para tarjeta SD con la función de añadir más

almacenamiento. Acceder como almacenamiento será capaz de copiar archivos dentro y fuera de la placa, si se crea un archivo de Python de nombre main.py como un archivo de la placa, MicroPython ejecutará ese archivo cuando se pone en marcha, lo que da como resultado que los comandos se ejecuten en la placa cuando se conecta a una PC.

Hardware.

El PyBoard tiene un microcontrolador STM32F405RG, con una CPU Cortex M4 de 168 MHz, 1024 Kb de memoria ROM y 192 Kb de RAM.

Así como también un puerto micro USB y una ranura para tarjeta microSD, la placa también cuenta con un acelerómetro de tres ejes, reloj en tiempo real, dos switches y cuatro LED's (RGB).

Permite la conectividad con periféricos y otros componentes a través de pines GPIO.

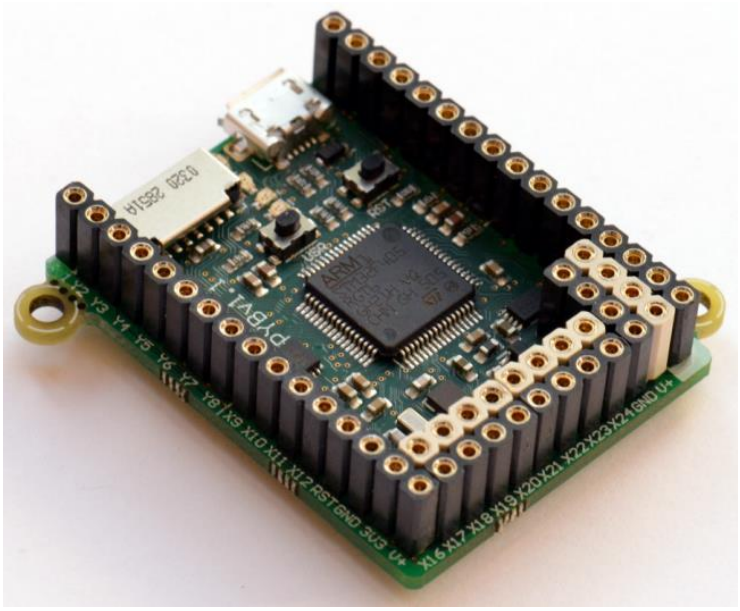

Figura 5. *La PyBoard original. (Tollervey, 2017)*

Configuración del entorno de desarrollo.

La PyBoard al aparecer como dispositivo removible USB en una PC, comenzará con cuatro archivos:

- boot.py

  Se ejecuta cuando el dispositivo inicia y establece varias configuraciones.
- main.py

  Es el Script principal que contiene el código, que se ejecuta inmediatamente después de boot.py.
- README.txt

  Contiene información acerca de PyBoard.
- pybcdc.inf

  Es un controlador de Windows para configurar el serial USB.

El módulo **pyb** es el que contiene todas las funciones y clases que se necesitan para trabajar con el hardware de PyBoard. (Tollervey, 2017)

GPIO

Entrada/Salida de Propósito General (GPIO) es como todos los dispositivos se conectan al mundo exterior. Esta conexión es lograda de una manera física a través de “pines” que se conectan al microcontrolador que MicroPython ejecuta.

Pines.

Un pin es un área conductora conectada al microcontrolador a través de los cuales existe una comunicación con el exterior y periféricos.

Los pines se deben nombrar para poder hacer referencia a ellos en el código, dichas referencias dependen de la versión de MicroPython que se está ejecutando, ya que se encuentran en diferentes lugares. Los PyBoard tienen una clase **Pin** que se instancia con el nombre del pin y algunas de sus características, por ejemplo, si se trata de una entrada o salida digital.

Los pines GPIO pueden estar en tres estados: alto, bajo y flotante. Al configurar el “pull” de un pin se evita que el estado de flotante indeterminado cuya señal reflejará las condiciones eléctricas ambientales del pin.

UART

Cuando una placa se conecta en una computadora vía cable USB es posible comunicarse con el dispositivo usando REPL, lo cual es posible por el *universal asynchronous receiver/transmitter* (UART), una parte del microcontrolador esta entre la comunicación serial y paralela. Mensajes seriales vienen de un bit a la vez, y el hardware del UART une la señal en bytes (representación paralela usualmente de 8 bits) que son enviados por un bus interno para ser procesado por un microprocesador.

Para realizar la transferencia, es necesario realizar algunos arreglos. **Primero**, el puerto de transmisión (llamado TX) de un dispositivo A debe conectarse al puerto de recepción (llamado RX) de un dispositivo B, y viceversa.

Segundo, es necesario estar de acuerdo con la sincronización de la comunicación serial para que el UART pueda detectar señales alto/bajo individuales, esto es la velocidad de comunicación y es expresada en uno de los estándares de tasas de baudios: 9600, 14400, 19200, 28800, 38400, 57600, y 115200 bits por segundo.

Tercero, algunas veces es necesario especificar el número de bits por byte, el estándar suele ser 8 bits; también se necesita especificar si se quiere un bit de paridad, el cual tiene la función de detectar errores en la transmisión; y el número de bits de parada que indican el final de una transmisión.

El UART también tiene una cola FIFO (primero en entrar/primero en salir) para que los bytes se almacenen en el búfer si no se leen tan pronto como se reciben. El UART en las placas con MicroPython está conectado a los pines internos USB-UART TX/RX que lo conectan a un convertidor USB, permitiendo conectar el UART a una PC; por defecto la tasa de baudios para conectar MicroPython de esta manera es de 115200. (Tollervey, 2017)

Constructores en UART.

Construir un objeto UART en el bus dado. El valor del bus puede variar según los diferentes ámbitos. No es necesario agregar ningún parámetro adicional este inicializará el bus si se le da algún parámetro adicional.

```
class pyb.UART(bus, ...)
```

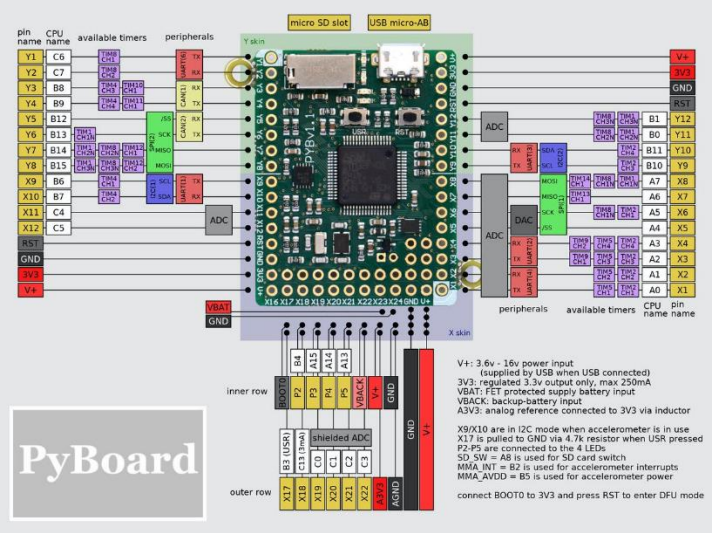

Figura 6. *Funcionamiento de los pines en el PyBoard. (iottrends.tech, 2020)*

Hay 6 interfaces UART presentes en el PyBoard. Los pines físicos de PyBoard en los autobuses UART son:

- **UART(4)** está en XA: (TX, RX) = (X1, X2) = (PA0, PA1)
- **UART(1)** está en XB: (TX, RX) = (X9, X10) = (PB6, PB7)
- **UART(6)** está en YA: (TX, RX) = (Y1, Y2) = (PC6, PC7)
- **UART(3)** está en YB: (TX, RX) = (Y9, Y10) = (PB10, PB11)
- **UART(2)** está en: (TX, RX) = (X3, X4) = (PA2, PA3) (iottrends.tech, 2020)

2.4. Python

Tkinter, Interfaz de Python para Tcl/Tk.

El paquete tkinter («interfaz Tk») es la interfaz por defecto de Python para el kit de herramientas de GUI Tk.

Arquitectura.

Tcl/Tk no es una biblioteca única debido a que consiste en unos pocos módulos distintivos entre sí, cada uno separados por funcionalidades y que poseen su documentación oficial.

- Tcl

  La librería TCL contiene una interfaz con C que crea y gestiona una o más instancias del intérprete de TCL, corre los comandos en TCL en esas instancias y agrega comandos personalizados tanto en Tcl o C. Cada intérprete tiene su cola de eventos y entrega facilidades para enviar procesos a la cola y procesarlos. A diferencia de Python, el modelo de ejecución en Tcl es diseñado alrededor de cooperar en múltiples asignaciones y Tkinter ayuda a salvar estas diferencias.

- Tk

  Es un paquete de Tcl package implementado en C el cual agrega comandos personalizados donde se pueden manipular widgets en la GUI. Cada objeto de la clase Tk agrega su instancia de interprete Tcl con Tk cargado en el. Los widgets de Tk son muy personalizables, aunque a costa de una apariencia anticuada. Tk usa eventos de la cola de Tcl para generar y procesar eventos de la GUI.

- Ttk

  Es el nuevo tema Tk (Ttk) es una nueva familia de widgets de Tk que proveen una mejor apariencia en diferentes plataformas más que varios de los widgets clásicos de Tk. Ttk es distribuido como parte de Tk estando disponible a partir de la versión 8.5.

Módulos de TKinter.

El soporte para TKinter se distribuye para varios módulos. La mayor parte de las aplicaciones necesitaran el módulo base TKinter, así como también el módulo tkinter.ttk, el cual entrega el conjunto de widgets temáticos y la API correspondiente:

```
from tkinter import *
from tkinter import ttk
clase tkinter.Tk(screenName=None, baseName=None, className='Tk',
useTk=1)
```

La clase Tk se instancia sin argumentos. Esto crea un widget de nivel superior de Tk que generalmente es la ventana principal de una aplicación. Cada instancia tiene su propio intérprete Tcl asociado.

```
Tkinter.Tcl(screenName=None, baseName=None, className='Tk', useTk=0)
```

La función Tcl() es una función de fábrica que crea un objeto muy similar al creado por la clase Tk, excepto que no inicializa el subsistema Tk. (Python Software Foundation, 2022)

Φ Para más información ver el Anexo, **Declaración de algunos Widgets en interfaz TKinter.**

Comunicación serial en Python

```
clase serial. Serial

__init__(port=None, baudrate=9600, bytesize=EIGHTBITS,
parity=PARITY_NONE, stopbits=STOPBITS_ONE, timeout=None, xonxoff=False,
rtscts=False, write_timeout=None, dsrdtr=False,
inter_byte_timeout=None)
```

Parámetros	
	• **port:** nombre del dispositivo o **None**. • **baudrate** (int) – Velocidad de baudios como 9600 o 115200, etc. • **bytesize** – Número de bits de datos. Valores posibles: FIVEBITS, SIXBITS, SEVENBITS, EIGHTBITS • **parity**: habilita la comprobación de paridad. Valores posibles: PARITY_NONE, PARITY_EVEN, PARITY_MARK PARITY_ODD, PARITY_SPACE • **stopbits** – Número de bits de parada. Valores posibles: STOPBITS_ONE, STOPBITS_ONE_POINT_FIVE, STOPBITS_TWO • **timeout** (float): establece un valor de tiempo de espera de lectura. • **xonxoff** (bool): habilita el control de flujo de software. • **rtscts** (bool): habilita el control de flujo de hardware (RTS/CTS). • **dsrdtr** (bool): habilita el control de flujo de hardware (DSR / DTR). • **write_timeout** (float): establece un valor de tiempo de espera de escritura. • **inter_byte_timeout** (float): tiempo de espera entre caracteres, Ninguno para deshabilitar (predeterminado).

Excepciones	• **ValueError**: se elevará cuando los parámetros estén fuera del rango, por ejemplo, velocidad en baudios, bits de datos. • **SerialException** – En caso de que el dispositivo no se pueda encontrar o no se pueda configurar.

Tabla 1. *Tabla de los parámetros que se pueden especificar en la clase Serial de Python.*

El puerto se abre inmediatamente en la creación del objeto, cuando se da un puerto. No se abre cuando el puerto es **None** y se requiere una llamada sucesiva a **open()**.

Φ Para más información ver Anexo**, Ejemplo de comunicación serial entre Python y MicroPython.**

2.5. Ophyra

Es una plataforma de desarrollo concebida para proyectos electrónicos de alto desempeño, diseñada y construida por Intesc Electronics & Embedded. Está diseñada con el microcontrolador STM32F407VGT6 de STMicroelectronics el cual está integrado por un ARM Cortex-M4®.

El microcontrolador de Ophyra cuenta con una Unidad de Punto Flotante (FPU) e integra un set de instrucciones orientados a DSP (Procesador de Señales Digitales). La velocidad del reloj principal de Ophyra es 168Mhz y además cuenta con una serie de dispositivos periféricos que permiten explorar todo el entorno de procesamiento del microcontrolador.

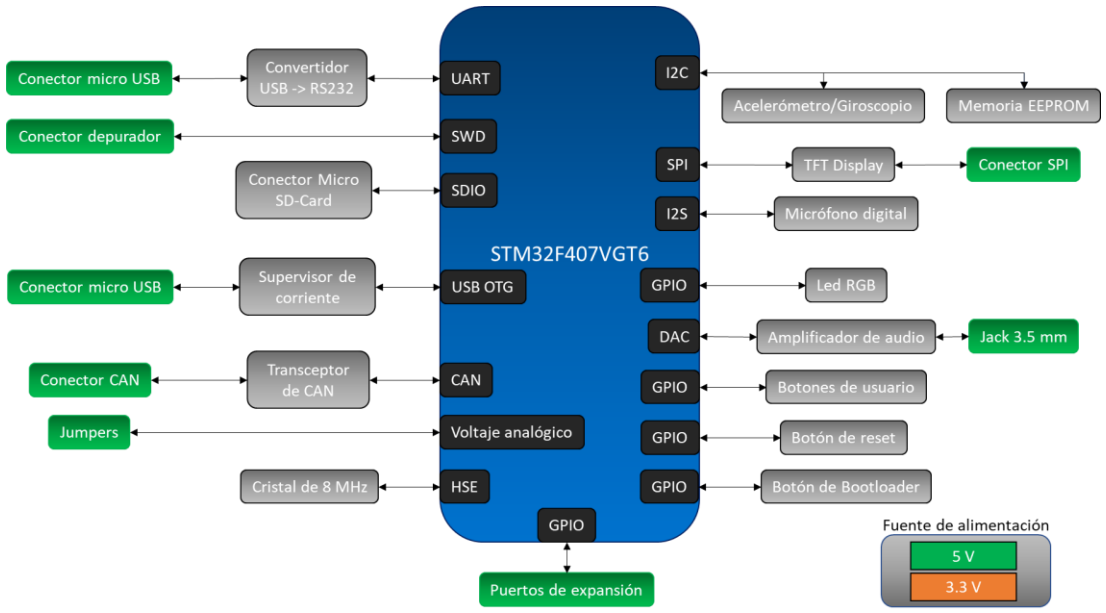

Figura 7. *Diagrama a bloques del sistema Ophyra. (Intesc Electronics & Embedded, 2017)*

Puertos de propósito general y su configuración.

Cada uno de los pines GPIO puede ser configurado por software como salida (push-pull, open-drain, pull-up o pull-down), como entrada (flotante, pull-up o pull-down) o como función alternativa periférica por lo que la mayoría de los GPIO se comparten con dispositivos digitales, analógicos o funciones alternativas; los cuales son de alta corriente (20mA) y tienen selección de velocidad para manejar el ruido interno, el consumo de energía y las emisiones electromagnéticas.

La mayoría de los GPIO son tolerantes a 5v, excepto si están en modo analógico o sí son pines de oscilador, que en estos casos se recomienda un voltaje de entrada de 3.3v como máximo. (Intesc Electronics & Embedded, 2017)

Bus de datos USB-OTG.

Se trata de una extensión del USB 2.0, que permite a los dispositivos con conectores USB, más flexibilidad en la gestión de dicha conexión. Los dispositivos compatibles con OTG son capaces de abrir una sesión, controlar la conexión e intercambiar las funciones

maestro/esclavo; ya sea una cámara digital, un pendrive, un módem USB, un teclado, un ratón, etc.

Por ello, la norma USB-OTG introduce dos nuevos protocolos: El protocolo SRP (Session Request Protocol: protocolo de solicitud de sesión); y el protocolo HNP (Host Negotiation Protocol: protocolo de negociación de host).

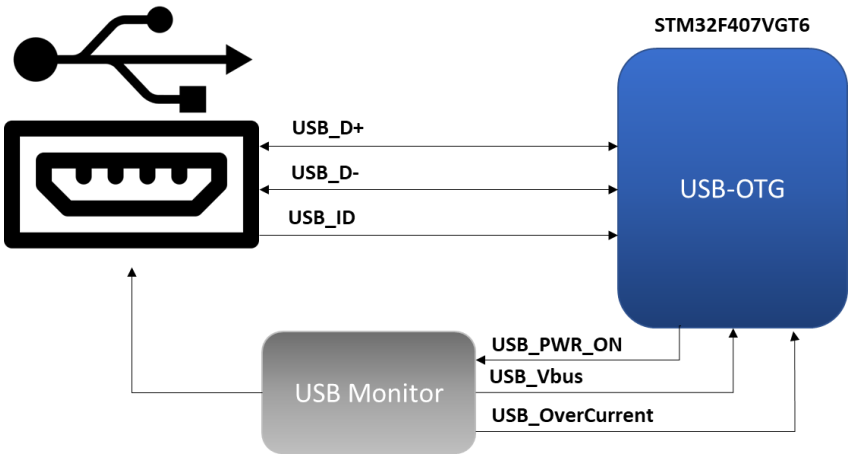

Figura 8. *Diagrama a bloques del bus de datos USB-OTG. (Intesc Electronics & Embedded, 2017)*

Nombre	GPIO	Función	Descripción
USB_D+	PA12	USB_OTG_FS_DP	Señal positiva de datos.
USB_D-	PA11	USB_OTG_FS_DM	Señal negativa de datos.
USB_ID	PA10	USB_OTG_FS_ID	Identificación del rol del dispositivo.
USB_PWR_ON	PC7	GPIO_ouput	Habilitación de energía en el Puerto USB.
USB_Vbus	PA9	USB_OTG_FS_VBUS	Identificación de voltaje en el puerto USB.
USB_OverCurrent	PA8	GPIO_input	Detección de consumo mayor a 500mA en el puerto USB.

Tabla 2. *Disposición de los pines para el bus de datos USB-OTG. (Intesc Electronics & Embedded, 2017)*

Bus de datos del puerto COM.

Para el intercambio de información se tienen dos alternativas:

- Transmisión en paralelo: usando un número dado de líneas de comunicación igual al tamaño de cada palabra de datos.
- Transmisión en serie: transmitir cada bit que constituye nuestra palabra uno por uno.

La flexibilidad de este puerto aumenta debido a que Ophyra cuenta con un transceptor FTDI que facilita la conexión a un puerto USB de la PC.

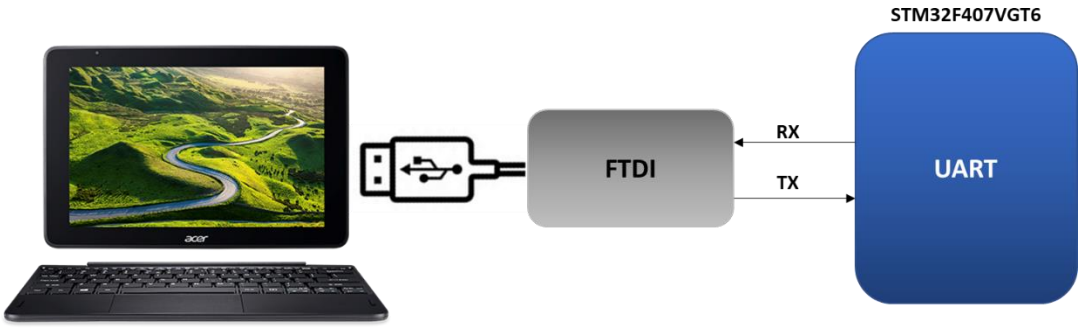

Figura 9. *Diagrama a bloques del puerto COM. (Intesc Electronics & Embedded, 2017)*

Nombre	GPIO	Función	Descripción
COM_TXD	PB11	USART3_TX	Línea de transmisión de datos
COM_RXD	PB10	USART3_TX	Línea de recepción de datos

Tabla 3. *Disposición de los pines del puerto COM. (Intesc Electronics & Embedded, 2017)*

Pulsadores y Led RGB.

Ophyra cuenta con seis pulsadores, de los cuales Reset y Boot son dedicados para el sistema. Los cuatro pulsadores restantes son de propósito general para el usuario, dichos pulsadores son de configuración normalmente abiertos.

También cuenta con un led RGB de propósito general, conectado al microcontrolador en configuración de ánodo común.

Nombre	GPIO	Función	Descripción
Reset	NRST	N/A	Reinicia el microcontrolador
Boot	Boot0	N/A	Forza al microcontrolador a entrar en modo Bootloader; si se mantiene presionado durante un reinicio.
SW0	PC2	GPIO_Input	Detecta el nivel lógico del botón sW0
SW1	PD5	GPIO_Input	Detecta el nivel lógico del botón sW1
SW2	PD4	GPIO_Input	Detecta el nivel lógico del botón sW2
SW3	PD3	GPIO_Input	Detecta el nivel lógico del botón sW3
Led Rojo	PE0	GPIO_Output	Controla el encendido del led rojo
Led Verde	PE1	GPIO_Output	Controla el encendido del led verde
Led Azul	PE2	GPIO_Output	Controla el encendido del led azul

Tabla 4. *Disposición de los pines de los pulsadores y el led RGB. (Intesc Electronics & Embedded, 2017)*

2.6. Instrucciones de un Controlador Lógico Programable

Funciones lógicas.

Este subconjunto de instrucciones define, en el lenguaje empleado, los operadores lógicos booleanos: Y (AND), O (OR), y complemento.

Estas instrucciones permiten la resolución de los tratamientos de información correspondientes a procesos lógicos secuenciales, mediante la implementación de las funciones lógicas o ecuaciones de Boole. (Badía, 1998)

Realizan una operación lógica entre dos combinaciones binarias. Los operandos son los contenidos de los acumuladores ACU1 y ACU2, y la operación lógica se realiza bit a bit entre todos los bits de ambos. El resultado de la operación lógica se almacena en el ACU1 y el contenido del ACU2 no se modifica. (Pérez, 2009)

$$ACU1[operación\ logica]ACU2 = ACU1$$

Temporización.

Se trata de una función temporal en la que una variable temporizada, adquiere el estado de una variable de control (estado 0 o 1) transcurrido un tiempo ***t*** preestablecido.

Cuando la variable temporizada adquiere el estado 1 de la variable de control con un retardo *t*, se trata de un temporizador con retardo a la activación; mientras que, si partiendo del estado 1 la variable temporizada alcanza el estado 0 de la variable de control con un retardo ***t***, se trata de un temporizador con retardo a la desactivación. (Badía, 1998)

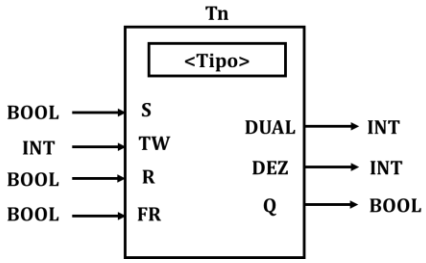

Figura 10. *Símbolo lógico de un temporizador. (Pérez, 2009)*

La función de cada una de las variables de entrada y de salida es la siguiente:

S: Entrada de disparo o arranque (Trigger). Al aplicarle un flanco de subida (indicado mediante el símbolo hace que el temporizador comience a disminuir el tiempo que queda para finalizar la temporización, a partir del especificado en la entrada TW.

TW: Entrada del valor inicial de la temporización. Está asociada a la instrucción de carga L mediante la cual se introduce en el ACU1 el valor inicial de la temporización.

R: Entrada de borrado (Reset). Cuando está en nivel “1”, hace que la salida del temporizador permanezca en nivel “0”.

FR: Entrada de habilitación o desinhibición. Se utiliza para provocar el redisparo del temporizador.

Q: Variable lógica de salida. Indica el estado en el que se encuentra el temporizador. Su comportamiento depende del tipo de temporizador.

DUAL: Variable de salida. Es una combinación binaria que en cada instante indica, en binario natural, el tiempo que queda para que finalice la temporización.

DEZ: Variable de salida. Es una combinación binaria que en cada instante indica, en BCD natural, el tiempo que queda para que finalice la temporización

Existen cinco tipos de temporizadores:

Temporizador de impulso (SI)

En este tipo de temporizador, la variable de salida Q se activa y permanece activada como máximo durante el tiempo "t" especificado en la entrada TW a partir del instante en que se aplica un flanco de subida a su entrada de arranque S.

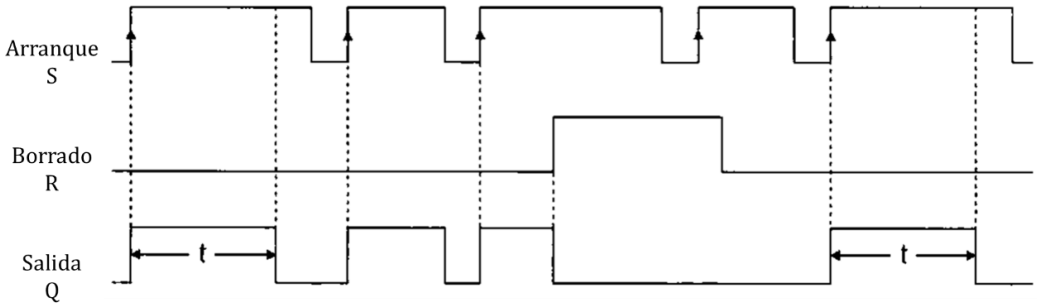

Figura 11. *Cronograma de un temporizador de impulso. (Pérez, 2009)*

Temporizador de impulso prolongado (SV)

El comportamiento de este tipo de temporizador coincide con el del temporizador SI, con la única diferencia de que, aunque la entrada de arranque se desactive antes de que haya transcurrido el tiempo "t" especificado en la entrada TW, la salida Q se mantiene activada hasta el final de dicho tiempo. Por ello también se le denomina temporizador con memoria.

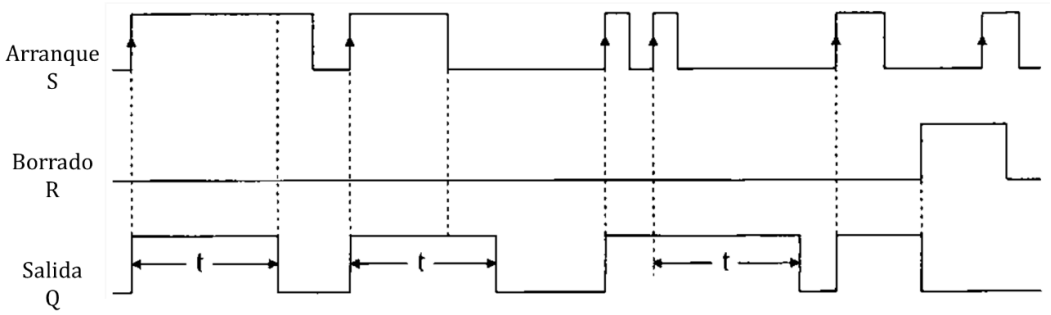

Figura 12. *Cronograma de un temporizador de impulso prolongado. (Pérez, 2009)*

Temporizador con retardo a la conexión (SE)

La variable de salida Q de este tipo de temporizador se activa después de transcurrido el tiempo “t” especificado en su entrada TW a partir del instante en que pasa de “0” a “1” la entrada de arranque S y permanece activada mientras ésta se encuentra en nivel uno.

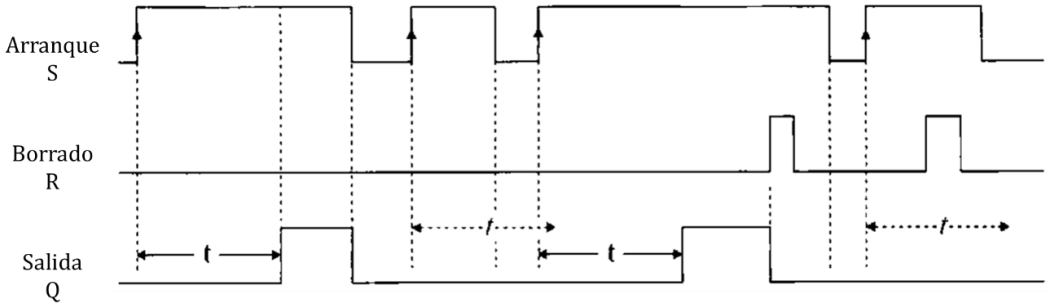

Figura 13. *Cronograma de un temporizador con retardo a la conexión. (Pérez, 2009)*

Temporizador de retardo a la conexión con memoria (SS)

El comportamiento de este tipo de temporizador es idéntico al del temporizador SE, con la única diferencia de que la temporización continúa, aunque el impulso aplicado a la variable de arranque tenga una duración inferior al tiempo “t” especificado en la entrada TW.

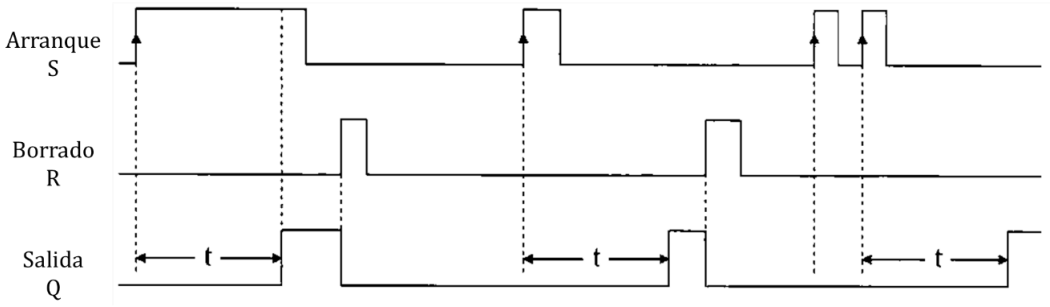

Figura 14. *Cronograma de un temporizador con retardo a la conexión con memoria. (Pérez, 2009)*

Temporizador de retardo a la desconexión (SA)

En este tipo de temporizador la variable de salida Q se activa al hacerlo la variable de arranque S y permanece activada hasta que transcurre el tiempo "t" especificado en la entrada TW, a partir del instante en el que la entrada de arranque pasa a nivel "0".

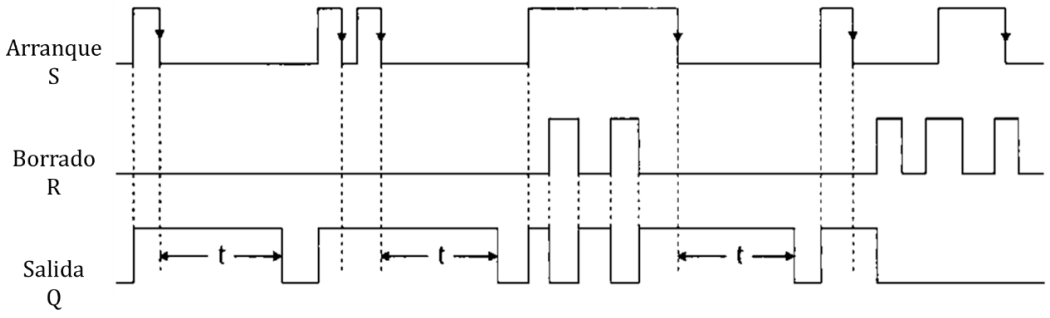

Figura 15. *Cronograma de un temporizador con retardo a la desconexión. (Pérez, 2009)*

Contaje.

Es una función en la que la variable de salida **Y** adquiere el estado 1, cuando han tenido lugar **n** transiciones de 0 a 1 del estado de la variable de contaje **X**; para que las transiciones sean contabilizadas, es preciso que la variable de control **V** (validación y puesta a cero) permanezca en el estado 1. (Badía, 1998)

Son instrucciones que hacen que el autómata programable cuente los flancos de subida de los impulsos que se produzcan en una determinada variable lógica. El contaje se puede realizar en sentido ascendente, en sentido descendente o en ambos sentidos.

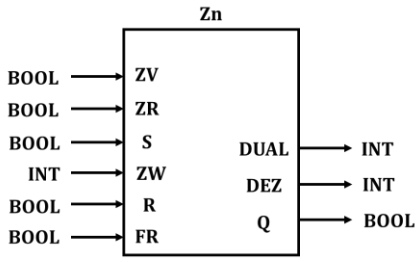

Figura 16. *Símbolo lógico de un temporizador. (Pérez, 2009)*

Las entradas y salidas del contador realizan las siguientes funciones:

ZV: Entrada de contaje ascendente. El contador incrementa su contenido o valor en una unidad cada vez que detecta un flanco en ella.

ZR: Entrada de contaje descendente. El contador decrementa su contenido en una unidad cada vez que detecta un flanco en ella.

S: Entrada de puesta a un valor inicial. Al aplicar un flanco de subida en esta entrada se introduce en el contador la combinación binaria especificada en la entrada ZW.

ZW: Entrada en la que se especifica la combinación binaria que se introduce en el contador cuando se aplica un flanco de subida a la entrada S.

R: Entrada de puesta a cero (Reset). Al aplicar un “1” en esta entrada se pone a cero el valor del contador.

FR: Entrada de habilitación o desinhibición. Se utiliza para hacer que el contador cuente un impulso, aunque este no se aplique a una de sus entradas de contaje.

Q: Variable lógica de salida. Indica el estado en el que se encuentra el contador.

DUAL: Variable de salida. Es una combinación binaria que indica en cada instante el contenido del contador en binario natural.

DEZ: Variable de salida. Es una combinación binaria que indica en cada instante el contenido del contador en BCD natural.

Capítulo 3. Desarrollo.

3.1. Arquitectura del sistema.

Realizar las conexiones necesarias que permiten la comunicación y programación del microcontrolador con ayuda de los softwares PyCharm como editor de código Python y Thonny para programar Micropython, se trata de al menos dos conexiones USB; la USB-OTG que permite el acceso a los archivos del microcontrolador y su programación de este, y la conexión del convertidor RS232 para la comunicación serial entre la PC y el puerto UART lo que posibilita el envío de información.

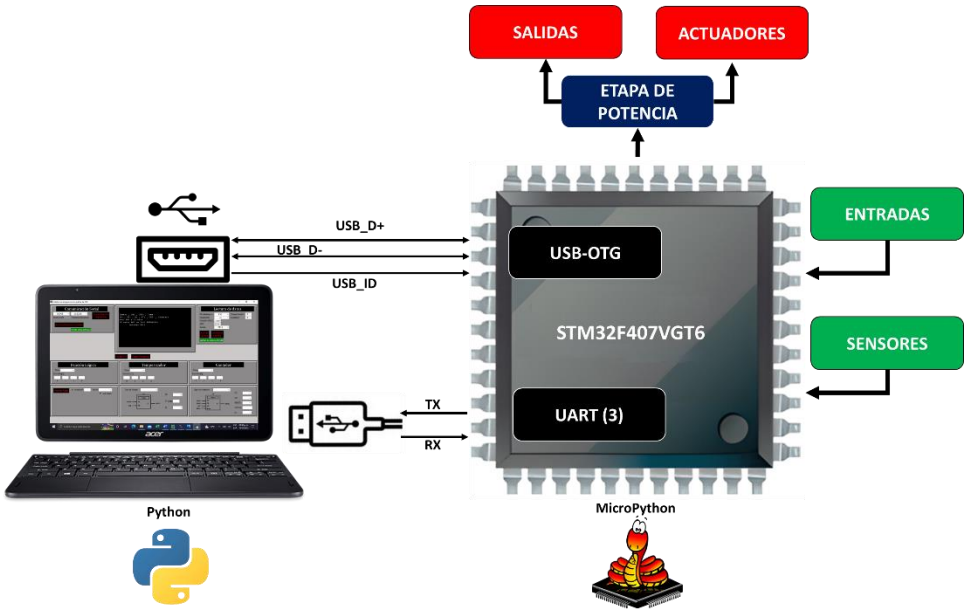

Figura 17. *Esquema general de conexiones entre una PC y un STM32.*

3.2. Lectura de entradas y salidas.

En la interfaz se define en primer lugar que entradas van a actuar en las instrucciones a implementar y la salida o resultado de estas; las cuales son elegidas mediante la escritura de los nombres o denominaciones de los pines a utilizar.

Para un mejor orden en la selección de entradas y salidas se establecieron de la siguiente manera:

Entradas. El uso de la librería ***pyb*** en Micropython tiene un método de lectura para determinar que pines son entradas y salidas mediante una cadena de 3 a 4 caracteres

conformado por la letra "**P**" de pin más el "**puerto**" más el/los "**número(s)**" del pin de dicho puerto. De acuerdo con esos tres elementos es como se forman los pines a seleccionar como entradas, por ejemplo:

Entradas: 3

$$Pin(0) = P + D + 0 = PD0$$
$$Pin(1) = P + D + 1 = PD1$$
$$Pin(2) = P + D + 2 = PD2$$

El valor de las entradas es seleccionado por una lista desplegable con los valores: 1 entrada, 2 entradas, 3 entradas, etc. y por medio de unas cajas de texto se introducen los pines como en las fórmulas anteriores.

Salidas. Estas variables se tratan como una constante, gracias a eso un puerto puede ser empleado únicamente para salidas.

Al tratarse de tres elementos de control (condición, temporizador, contador), existen tres selecciones de entradas para cada uno.

Como tal estos valores no se envían a través del serial solo son para su misma manipulación dentro de la interfaz gráfica, solo un botón actualiza las entradas que están declaradas en los cuadros de texto.

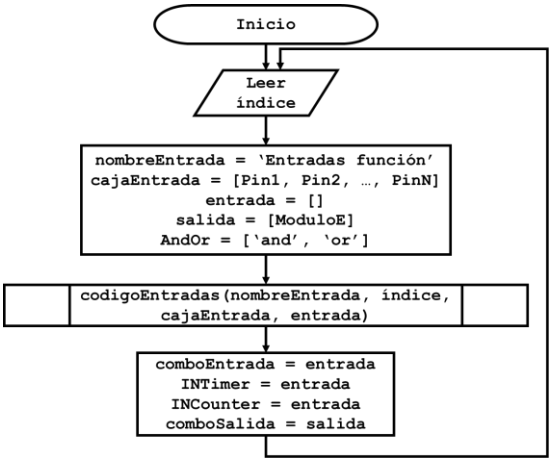

Figura 18. *Diagrama de flujo para la Lectura de las entradas en la Interfaz.*

Para más información ver el Anexo, **"Código de Lectura de entradas en la interfaz".**

3.3. Implementación de condicional if.

Lectura en la interfaz.

Una vez que las entradas ya están definidas en la sección de entradas para la condición nuevas variables aparecen para ser manipuladas, una condición se basa en ***n*** número de variables y ***n-1*** número de condiciones:

$$if\ \boldsymbol{var1}\ condición1\ \boldsymbol{var2}\ condición3\ \boldsymbol{var3}\ \ldots condición(n-1)\ \boldsymbol{var}(n)$$

Con ayuda de la fórmula anterior se genera la función en la interfaz por lo que solo es necesario recibir un número que represente a ***n***, el cual es leído por un cuadro de texto.

Una vez que la función es generada, como las entradas ya están definidas y debe ser capaz de utilizar cualquiera de las entradas declaradas, se usan listas desplegables que contengan todas las entradas; para la selección de la condición también se usan listas desplegables con las palabras **and-or**.

Otro parámetro a considerar en la función es la lógica negativa y positiva, mediante unas listas desplegables adicionales que contienen **1-0** donde '1' significa que es una entrada sin negación (Normalmente abierta) y el uso de '0' indica una entrada con negación (Normalmente cerrada), estas listas están por encima de cada entrada.

En cuanto a la salida, solo está limitada a una que es seleccionada mediante lista desplegable para seleccionar cualquiera que esté disponible.

Formato del envío de la función lógica. Esta parte se opta por dividirla en tres secciones de datos, uno para las entradas o pines, otro para las condiciones y uno más para la lógica negativa que le de entender al microcontrolador que primero decodifique las entradas, luego las condiciones y por último los estados de las entradas.

Primero enviar la palabra 'entradas' para la recepción de estas y después algunos parámetros como que salida se quiere ya que se trata de una constante y solo se envía el número de salida; después el microcontrolador estará listo la para la recepción de entradas:

```
X = ['entradas', número de Entradas, # de salida, pin de reset, act/desc, x variable]
```

La función lógica como extra puede ser usada como un set/reset con los parámetros “pin de reset” y “act/desc”, está ultima sirve como habilitación para determinar el modo de la función lógica, normal o set/reset.

Segundo, enviar otra palabra ‘condicion’ para comenzar la recepción de las condiciones más otros parámetros para después enviar las condiciones que se ocupan en la función lógica, esas condiciones están en 0 para AND y 1 para OR.

```
Y = ['condicion', número de Entradas – 1, x variable]
```

Y tercero, otra palabra ‘estado’ para la recepción de la lógica de las entradas con otros parámetros adicionales como el mismo dato de número de entradas.

```
z = ['estado', número de Entradas, x variable]
```

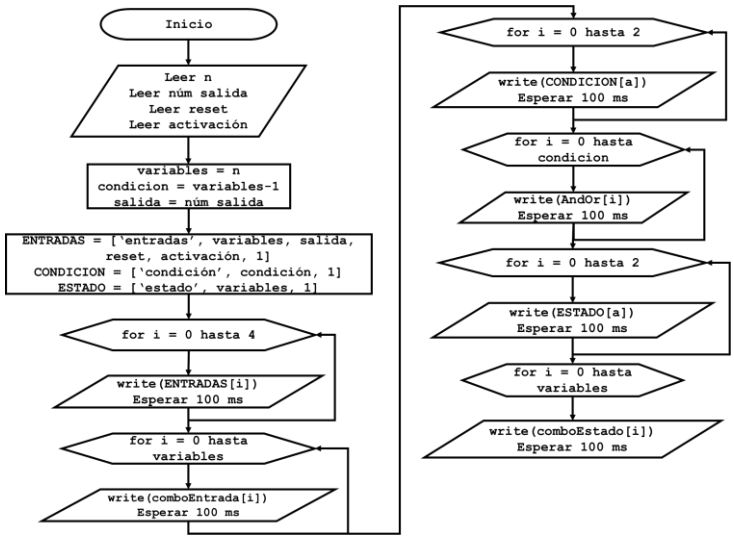

Figura 19. *Diagrama de flujo para el envío de los parámetros de la Función lógica.*

Lectura en MicroPython.

Los datos enviados por la interfaz al recibirlos el microcontrolador realizan la invocación cuatro funciones: `leerEntradas`, `leerCondiciones`, `leerEstados` y `evaluarRegla`; las cuales reciben los datos así como son enviados.

```
leerEntradas(número de Entradas, x variable)
```

El valor "**`x variable`**" sirve de apoyo para evitar que esa misma función se vuelva a ejecutar al momento de recibir las condiciones, está función almacena todos los pines a usar con la sintaxis de entrada en Micropython.

```
leerCondiciones(número de Entradas – 1, y variable)
```

De la misma manera "**`y variable`**" evita que se vuelva a ejecutar hasta que termine el proceso de recepción de la función lógica y es almacenada en un arreglo que contiene todas las condiciones (0's y 1's)

```
leerEstados(número de Entradas, z variable)
```

Como es un estado por cada variable, debe trabajar con el mismo número de entradas más una "**`z variable`**" que evita que reciba otros datos diferentes a 0's ó 1's.

```
respuesta = evaluarRegla(Entradas, Condiciones, Estados, número de Entradas – 1)
```

Gracias a que las Condiciones llegan en 0's ó 1's permite elegir fácilmente si se trata de una condición AND ó OR, y los estados de cada entrada son una simple igualación a 0 ó a 1, como resultado está función devuelve un valor en booleano (TRUE, FALSE) que manipula el estado de la variable de salida, básicamente para devolver ese valor booleano hace uso del método de acumuladores para evaluar una función n – 1 veces y acumular el estado lógico en una única variable, trabaja la función lógica en **mintérminos**.

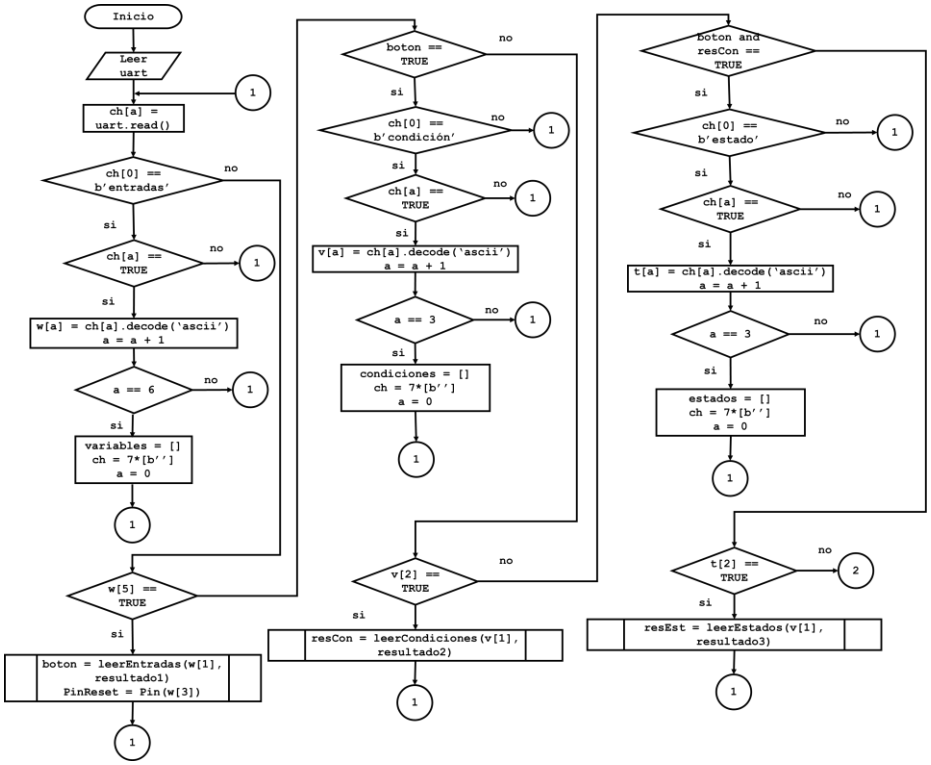

Figura 20. *Diagrama de flujo para la decodificación de los parámetros de la función lógica.*

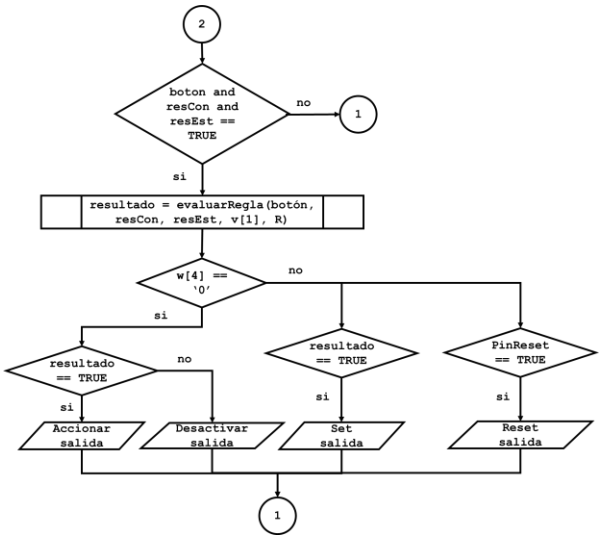

Figura 21. *Diagrama de flujo para la activación y desactivación de la salida.*

Para más información ver el Anexo, **Código de Envío y Recepción de la función lógica.**

3.4. Implementación del temporizador.

Lectura en la interfaz.

Con las entradas ya definidas en su respectiva sección, la lectura de cada parámetro del temporizador es mediante listas de desplegables para la activación y su salida, por otro lado, la lectura del tiempo es por un cuadro de texto.

La entrada solo está limitada a solo una, solo se acciona una salida y el tiempo es leído en formato de segundos.

La selección de que tipo de temporizador se quiere usar se encuentra regido por una lista de texto que contiene los tres tipos de temporización más comunes: temporizador a la conexión, a la desconexión y con impulso.

Formato de envío del temporizador. Este es enviado en un solo bloque de datos comenzado por la palabra '**`timer`**' para después enviar el resto de los parámetros, todos los datos son de tipo string.

```
X = ['timer', entrada del timer, salida del timer, tiempo, tipo de temporizador]
```

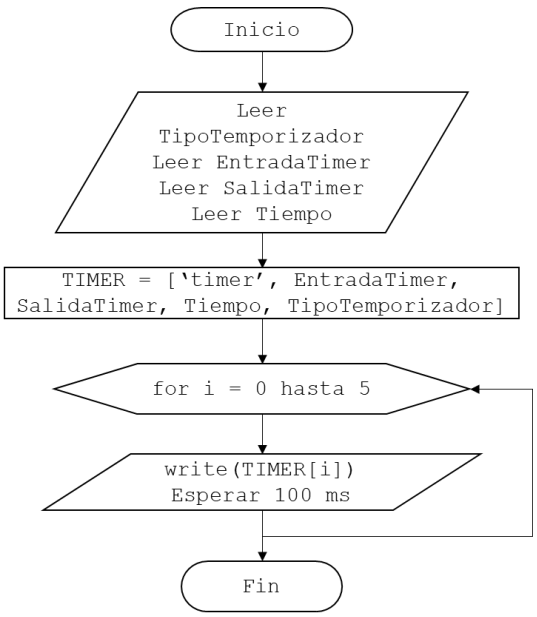

Figura 22. *Diagrama de flujo para el envío de los parámetros del Temporizador.*

Lectura en Micropython.

En el microcontrolador solo se hace uso de uno de sus Temporizadores.

Al leer los datos se la realiza la invocación de una solo función en la que a su vez invoca otras tres funciones de acuerdo con que tipo de temporizador se haya elegido en la interfaz.

```
temporizador(pin de entrada, pin de salida, tiempo, tipo de Temporizador)
```

Con un método de tipo condicional (if-elif-else) y la variable “tipo de Temporizador” se realiza la llamada de las otras tres funciones que solo están regidas por con la habilitación del temporizador de MicroPython y el periodo de este en milisegundos.

```
objetoTimer.init(period = # en milisegundos, callback = función a ejecutar)
```

Estos temporizadores usan segundos por lo que el periodo deberá ser de 1000. Cuando la función anterior realiza su invocación, se toman como bases los tres funcionamientos de cada tipo de temporizador:

Temporizador a la desconexión, se activa cuando un contador que empieza con el valor de “tiempo” disminuye en uno cada segundo hasta que es igual o menor a 0.

Temporizador a la conexión, se activa cuando un contador que inicia en 0 aumenta en uno cada segundo hasta que sea igual o mayor al valor de “tiempo”.

Temporizador con impulso, se activa cuando se el valor de la entrada del temporizador está en ALTO por una sola vez hasta que el botón se vuelva a presionar y haya dejado de contar.

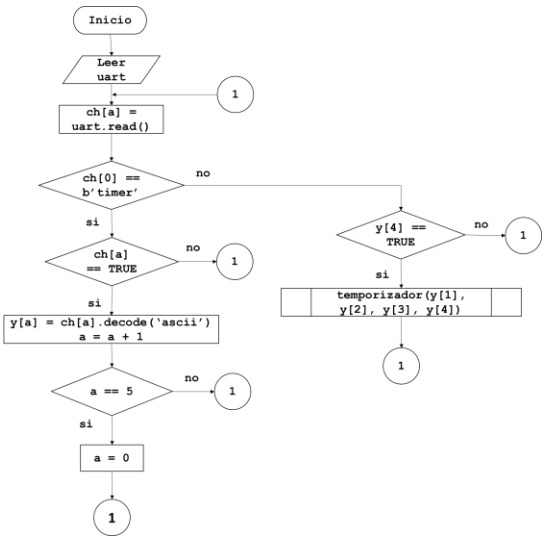

Figura 23. *Diagrama de flujo de la decodificación de los parámetros para el temporizador.*

Para más información ver el Anexo, **Código de envío y recepción del temporizador.**

3.5. Implementación del contador.

Lectura en la interfaz.

El contador en la interfaz solo tiene dos entradas para manejar los tres tipos de contador ya que el contador CTUD es capaz de incrementar y decrementar, por lo que se deben especificar esas dos entradas más el reset, todos los datos anteriores son introducidos por tres listas de textos para cada dato con los pines seleccionados para el contador, el número que rige la activación de la salida con una comparación con el conteo que se encuentra en ese momento se introduce como un número en formato de cadena y la salida nuevamente es una lista de textos.

Formato de envío.

De la misma manera que la lectura del temporizador, solo se envía la siguiente lista de datos con la palabra “counter” para dar a entender al Microcontrolador que va a recibir los datos de un contador.

```
X = ['counter', incremento, decremento, reset, salida Contador, número, tipo de Contador]
```

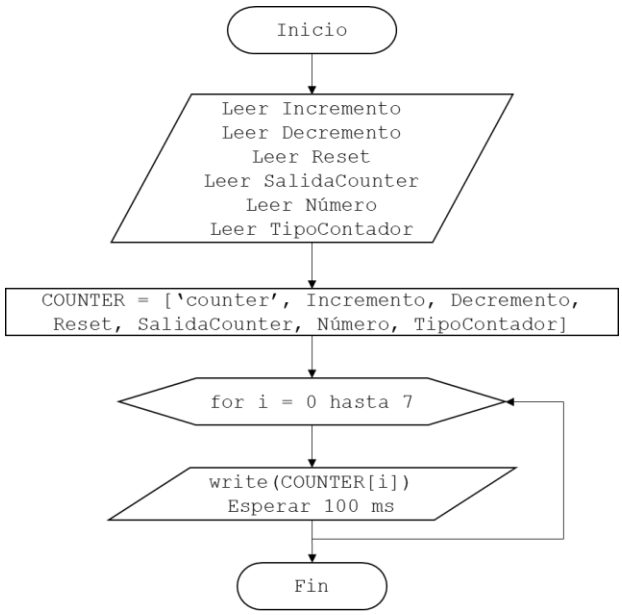

Figura 24. *Diagrama de flujo para el envío de los parámetros del Contador.*

Lectura en MicroPython.

Al recibir los datos del puerto serial, se derivan tres condiciones para elegir el tipo de contador que se va a usar en la programación, mediante la siguiente función:

```
contadores(Incremento, Decremento, Reset, Salida, numero, tipo Contador)
```

Con ayuda de la variable **"tipo Contador"** más una condición (if-elif-else), se ejecuta uno de los tres contadores, la activación por flanco de subida se logra gracias a la siguiente instrucción:

```
if Estado de un Pin == (ALTO o BAJO) and letra == 0:
   letra = 1
   # Operaciones a realizar al detectar flanco de subida
elif Estado de un Pin == (BAJO o ALTO) and letra == 1:
   letra = 0
```

Lo anterior permite hacer **n** instrucciones que se ven accionadas solo por un flanco de subida.

Al tratarse de los contadores tradicionales, el incremento solo aumenta en una unidad y de manera contraria, el decremento disminuye una unidad; cada contador tiene su

condición de activación de la salida mediante una comparación (<, >, <=, >=) al cumplirse que el valor del contador cumpla con dicha condición.

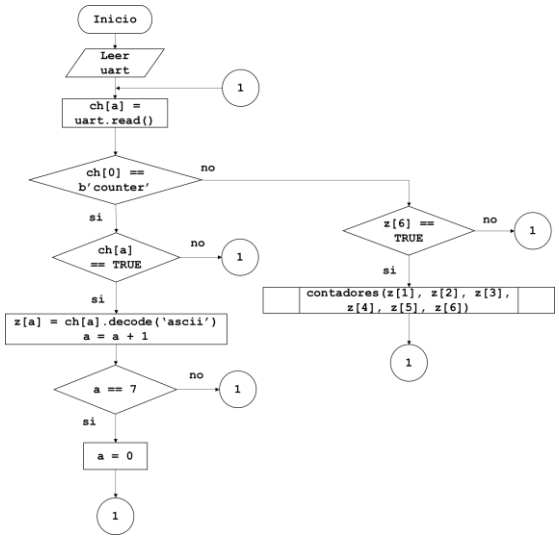

Figura 25. *Diagrama de flujo de la decodificación de los parámetros del Contador.*

Para más información ver el Anexo **Código de envío y recepción de datos del Contador.**

3.6. Implementación de lectura analógica y recepción de datos.

Lectura en la interfaz.

Al tratarse de una lectura analógica debe ocupar los pines que sean capaces de trabajar en modo analógico, comúnmente el puerto A es el que está adecuado a la Conversión Analógico-Digital (ADC).

La sección tiene una lista desplegable en la que se encuentran pines del puerto A. Para trabajar con el valor actual del ADC se requiere de una condición y un valor de referencia para manipular una señal de salida por lo que las condiciones más comunes son mayor igual y menor igual.

La condición es seleccionada por una lista, el valor de referencia se introduce como un valor entero que va de 0 a 4095, debido a que el microcontrolador STM32 tiene un ADC con resolución a 12 bits.

Formato de envío.

Para enviar el ADC se hace uso del siguiente arreglo:

```
ANALOGO = [‘analogo’, PinADC, condiciónADC, valorADC, salidaADC]
```

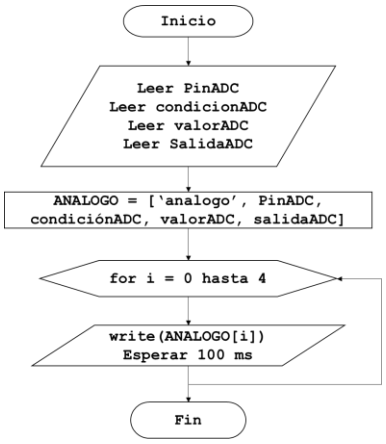

Figura 26. *Diagrama de flujo para el envío de los parámetros del ADC.*

La lectura de los datos son el valor en tiempo real del ADC de la tarjeta seleccionado en la interfaz y los valores actuales del temporizador y contador; tiempo y conteo respectivamente, todos esos valores son mostrados en cuadros de texto. La actualización de los datos es gracias a una invocación de una función que se ejecuta en pequeño intervalo de tiempo invocándose a sí misma.

Lectura en MicroPython.

Básicamente solo recibe los datos del puerto serial con la palabra ‘**analogo**’ para comenzar los demás parámetros y al reunirlos todos se llama una función para iniciar con la conversión analógico-digital.

```
analogico(Pin de entrada, condición, valor del ADC, Pin de salida)
```

La activación de la salida depende por el valor de la condición, puede ser un 0 o 1 para darle a entender que se activa la salida cuando el valor del Pin supere el valor de referencia o viceversa.

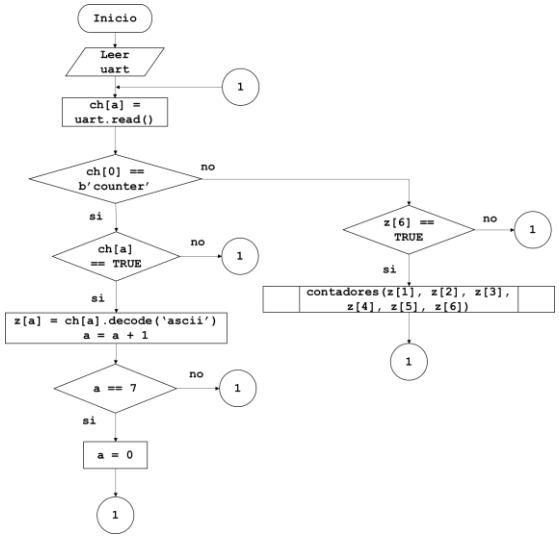

Figura 27. *Diagrama de flujo para el envío de parámetros del pin analógico.*

Activación y preparación para recepción de información en la interfaz.

Al tratarse una visualización en tiempo real, la información en el puerto serial debe estar fluyendo todo el tiempo por lo que se eligió una activación mediante un botón y su inhabilitación por otro botón que permita establecer a gusto del usuario el monitorear los valores de las instrucciones. Al no encontrar otro método de implementación esto puede alterar el funcionamiento de la interfaz por lo que si se quiere volver a cargar un programa ya no se debe estar compartiendo ningún dato.

Para una mejor entendimiento se muestra a continuación el diagrama de flujo para recepción de información en la interfaz.

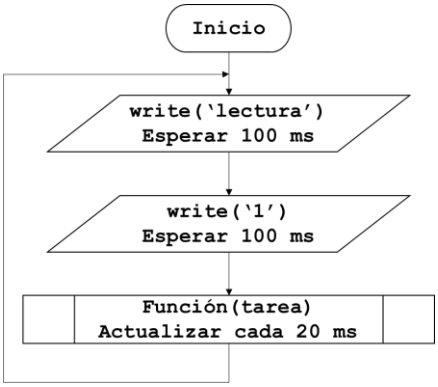

Figura 28. *Diagrama de flujo para la activación de la lectura de datos.*

Al presionar el botón “Activar Lectura” se envía la palabra ‘lectura’ y el valor ‘1’ que prepara al microcontrolador para hacer que envíe los datos del ADC, tiempo y conteo por el puerto serial e invoca un función que habilita una función que se ejecuta cada cierto tiempo actualizando valores en la interfaz.

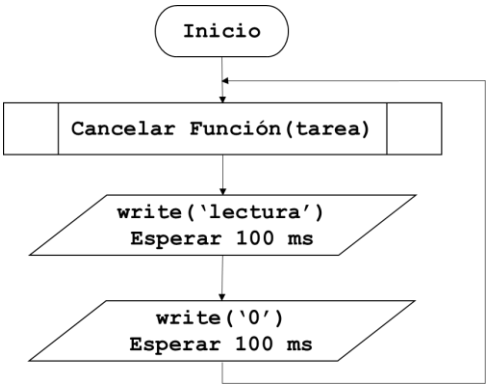

Figura 29. *Diagrama de flujo para la desactivación de la lectura de datos.*

Para finalizar al presionar el botón “Desactivar lectura” este cancela que se siga actualizando la interfaz, lo que también manda un ‘0’ para inhabilitar la transmisión de información en el microcontrolador, lo que permitirá programar otra vez el microcontrolador.

Capítulo 4. Resultados.

4.1. Uso y habilitación de la comunicación serial.

1. Presionar el botón “Buscar Puertos Disponibles”, ejecutará la función asociada a este para buscar los puertos COM que estén habilitados en una PC, si recién se habilita un puerto COM el volver a presionar el botón solo actualizará los puertos.

Figura 30. *Ubicación y selección del botón para buscar Puertos COM.*

2. Seleccionar el puerto COM en donde este conectada la tarjeta a programar, y seleccionar la velocidad en baudios apropiada y/o con el mismo valor con la que opera la tarjeta.

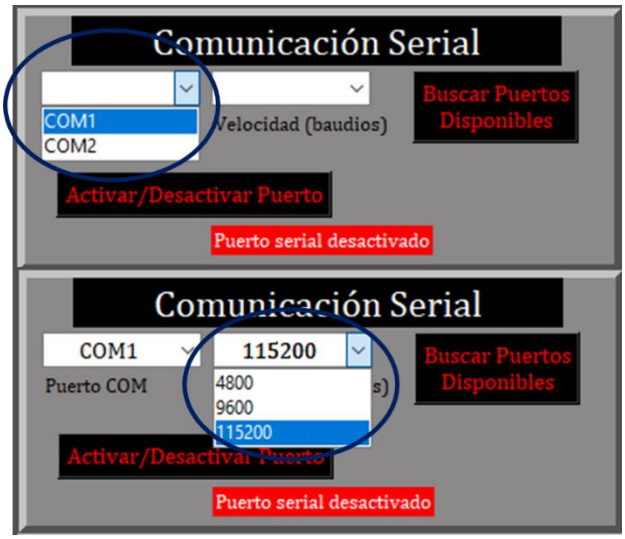

Figura 31. *Selección del Puerto COM (arriba), selección de la velocidad en baudios (abajo).*

3. Hacer uso del botón “Activar/Desactivar Puerto” que habilita la comunicación serial una vez que el puerto y la velocidad hayan sido especificadas, si se desea presionar antes de seleccionar el puerto y/o la velocidad, mandará un mensaje de error indicando que no se puede activar el Puerto COM.

Figura 32. *Mensaje al estar activada la comunicación serial.*

4.2. Declaración de entradas y salidas.

1. Seleccionar el número de entradas con las que se desea trabajar ya sea solo una o N entradas (la interfaz por el momento solo puede tener 4 por elemento de programación).

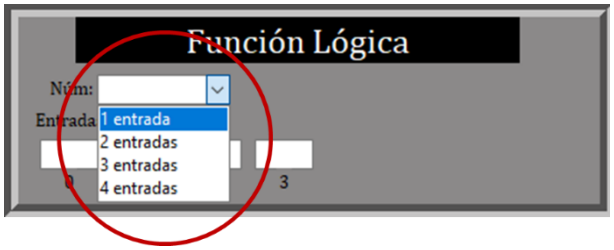

Figura 33. *Selección del número de entradas para la función lógica.*

2. Introducir los nombres de cada Pin a ocupar tomando en cuenta la estructura que maneja la tarjeta (ejemplo PA0), las cuales podrán ser escogidas en cada elemento de programación.

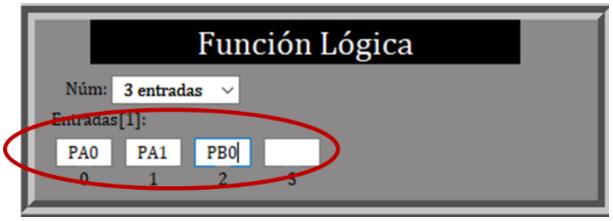

Figura 34. *Escritura en los cuadros de texto con el nombre de los Pines a utilizar.*

3. Repetir los pasos 1 y 2 para llenar las demás entradas del temporizador y el contador si es necesario.

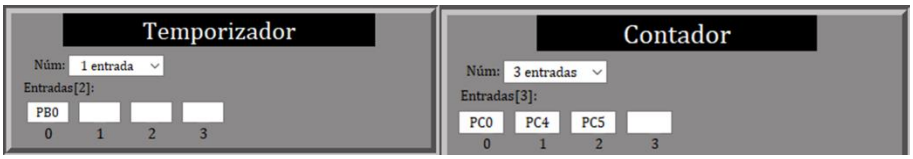

Figura 35. *Introducción de los Pines para el temporizador y el contador.*

4. Para subir las entradas para que puedan ser utilizadas solo basta con presionar el botón “Cargar E/S”, también funcionará como actualización de entradas por si cambia una. Se sugiere que para la regla lógica se declare una más a las variables a utilizar en la función lógica, el temporizador requiere solo una y el contador al menos tres entradas. Al cargar las entradas se mostrarán en la pantalla de la interfaz, las salidas serán por defecto el módulo E de la tarjeta.

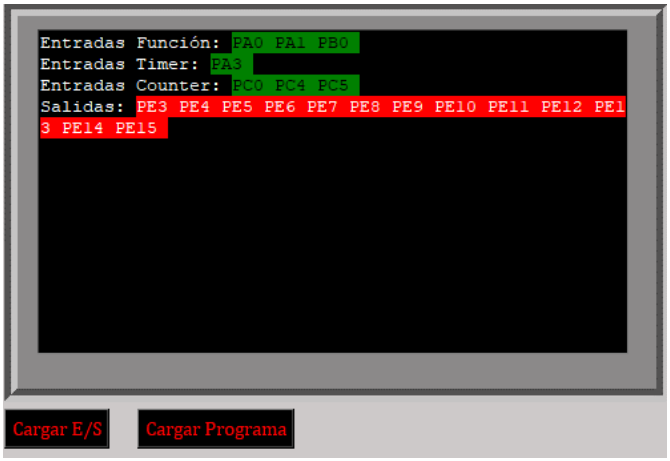

Figura 36. *Visualización de las entradas y salidas declaradas para cada uno de los elementos de programación.*

4.3. Uso de los elementos de programación.

Función lógica.

1. Definir de cuantas variables será la función lógica, si solo es una será una activación o desactivación directa, más de una ya permite seleccionar un conector lógico (and, or), una vez definido el número de variables solo faltará pulsar el botón “Generar Regla”.

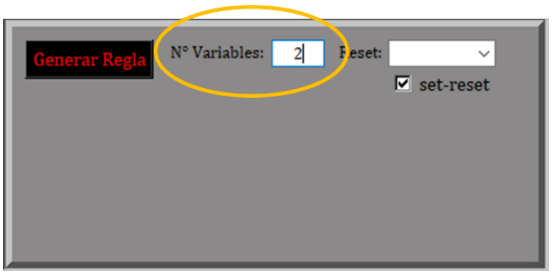

Figura 37. *Número de variables para la función lógica.*

El generar la regla invoca la estructura de una función lógica más un botón por si se quiere cambiar el número de variables de entrada, la función debe borrarse antes de generar otra función.

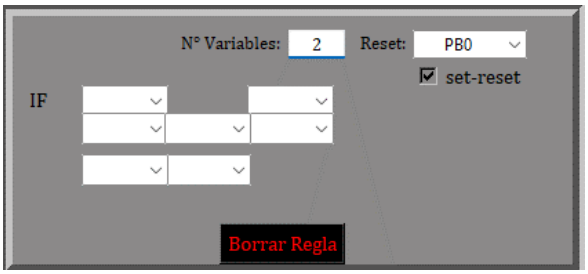

Figura 38. *Formato de la función lógica al usar el botón “Generar Regla”.*

2. Mediante las listas desplegables se seleccionan los pines que formarán parte de la función y el/los conectores lógicos que unen las variables, unas listas adicionales en la parte de arriba de cada variable indica si son entradas NC (0) o entradas NA (1).

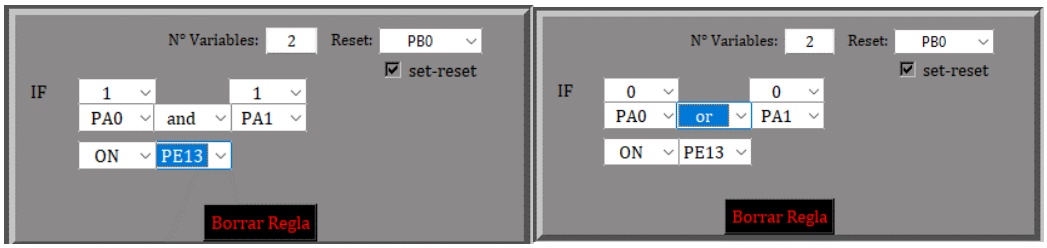

Figura 39. *Función con AND y entradas NA (izquierda), función con OR y entradas NC (derecha).*

3. Ya que la función ha sido declarada solo falta definir el pin de reset y cuenta con una casilla que representa el estado en que va a operar función lógica, como una activación simple o que almacene el bit de salida y sea borrado por pin especificado.

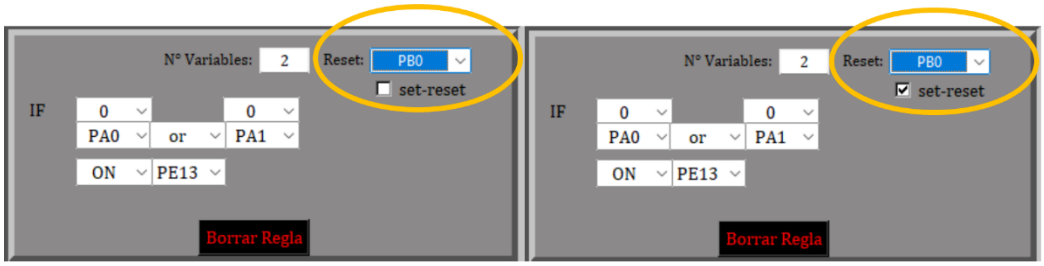

Figura 40. *Pin de reset desactivado (sin marcar casilla), Pin de reset activado (casilla marcada).*

Temporizador.

1. Primero será elegir el tipo de temporización que el programa vaya a utilizar, ya sea TON, TOF o impulso.

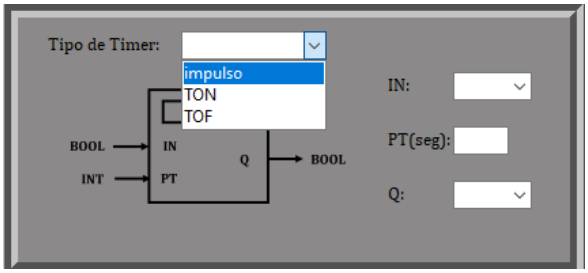

Figura 41. *Selección del tipo de temporizador a emplear.*

2. Segundo, seleccionar el pin de activación para la entrada del temporizador.

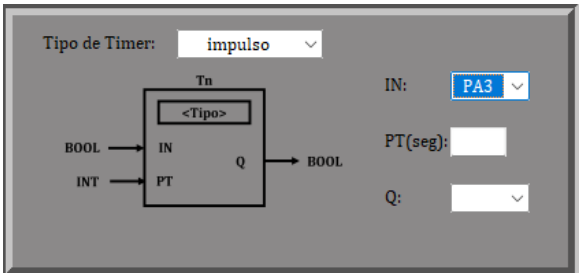

Figura 42. *Selección de la entrada del temporizador.*

3. Después introducir un número entero que representará los segundos que serán cargados en el temporizador, y seleccionar la salida Q.

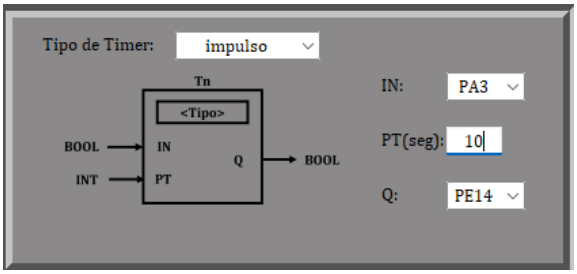

Figura 43. *Colocación del tiempo en segundos y selección del Pin de salida.*

Contador.

1. Seleccionar el tipo de contador, se encuentran CTU, CTD y CTUD.

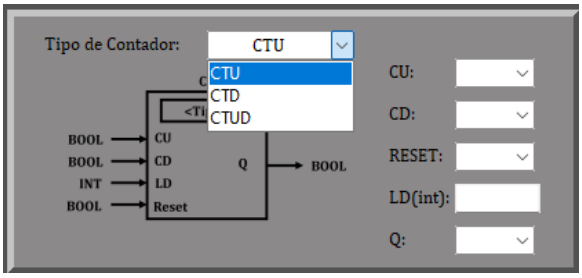

Figura 44. *Selección del tipo de Contador.*

2. Es necesario seleccionar los tres pines básicos para el funcionamiento de un contador, el de incremento, decremento y el reset.

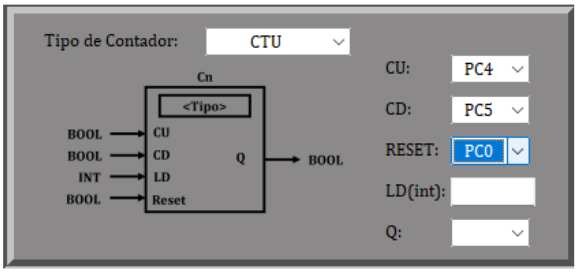

Figura 45. *Selección de los tres Pines de entrada para el incremento, decremento y reset.*

3. Colocar la carga o Load como un número entero que servirá de referencia en la activación de la salida; y escoger un pin de salida.

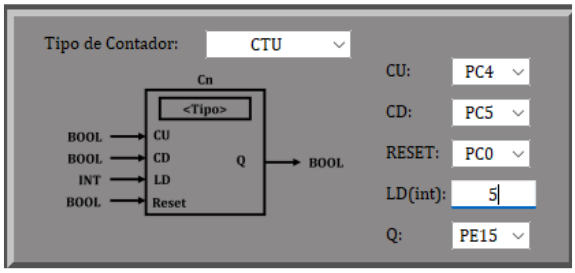

Figura 46. *Carga en número entero y el Pin de salida.*

Convertidor analógico-Digital.

1. Seleccionar el Pin analógico que se necesite usar.

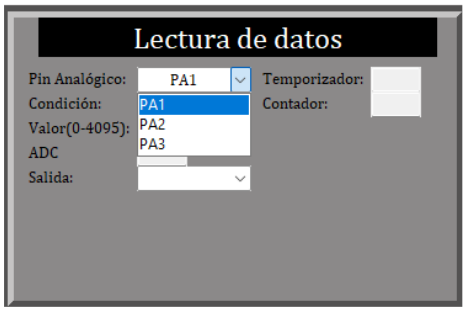

Figura 47. *Selección del Pin Analógico.*

2. Seleccionar la condición de la lista desplegable que contiene mayor que y menor que.

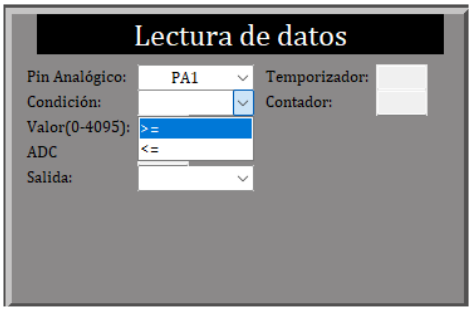

Figura 48. *Selección de la condición <= ó >=.*

3. Introducir el valor de referencia entre 0 y 4095, seleccionar el pin de salida, los cuadros de texto para temporizador, contador y ADC son habilitados hasta que el programa es cargado en el microcontrolador ya que esto habilita dos botones que permiten la recepción del dato analógico, el tiempo y el conteo en tiempo real. Si la recepción ya no se va a utilizar es necesario detener la lectura de los datos para volver a cargar otro programa.

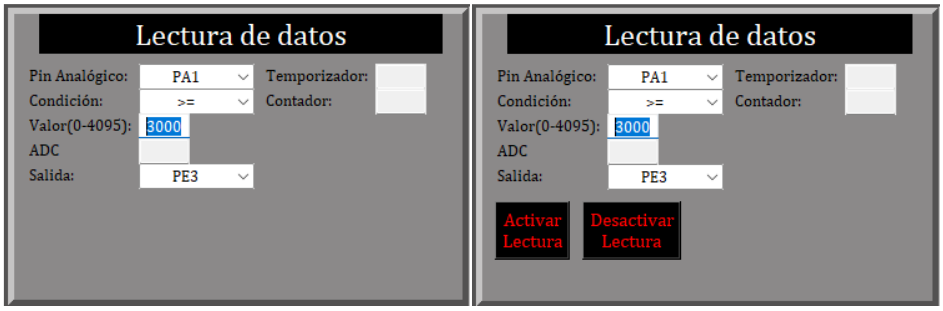

Figura 49. *Escritura del valor de referencia del ADC, activar salida cuando el valor real del ADC es mayor al de referencia (izquierda) y desactivar salida cuando el valor real del ADC es menor al de referencia (derecha).*

Al introducir y seleccionar todas las variables y pines que se necesiten en X aplicación solo falta cargar el programa con el botón "**cargar programa**".

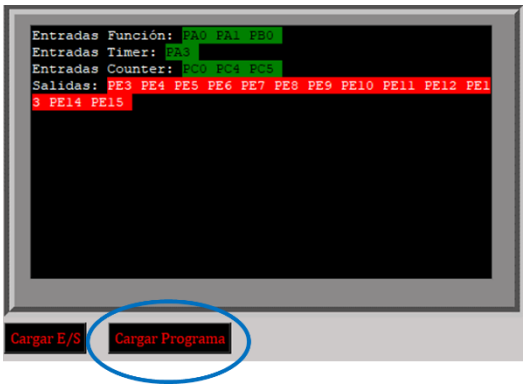

Figura 50. *Botón para cargar el programa al microcontrolador al terminar la programación.*

Al enviar todos los datos por el serial, la pantalla de pseudocódigo se actualizará mostrando todo lo que se encuentra cargado en el microcontrolador.

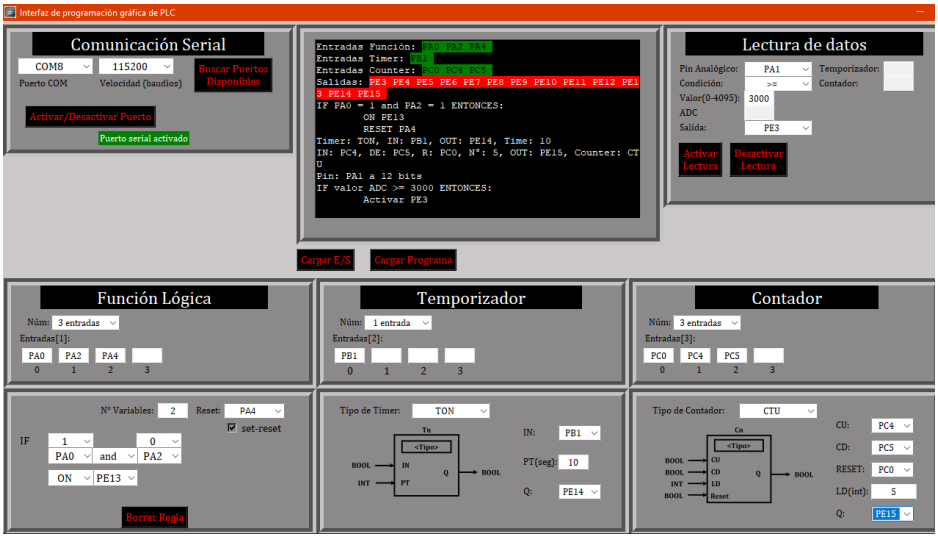

Figura 51. *Visualización del código al cargar la programación en el microcontrolador.*

4.4. Uso de la interfaz en diferentes computadoras con archivo exe.

Con la interfaz finalizada esta no puede quedar simplemente como un archivo de python ya que eso implicaría estar pasando el código a diferentes computadoras y estar instalando las librerías necesarias y software faltante dificultando su uso.

Por ello Python ofrece una librería que permite convertir cualquier archivo py (archivo de Python) en uno ejecutable y este puede ser único o con archivos complementarios.

En una computadora de 32 bits marca ACER® sistema operativo Windows:

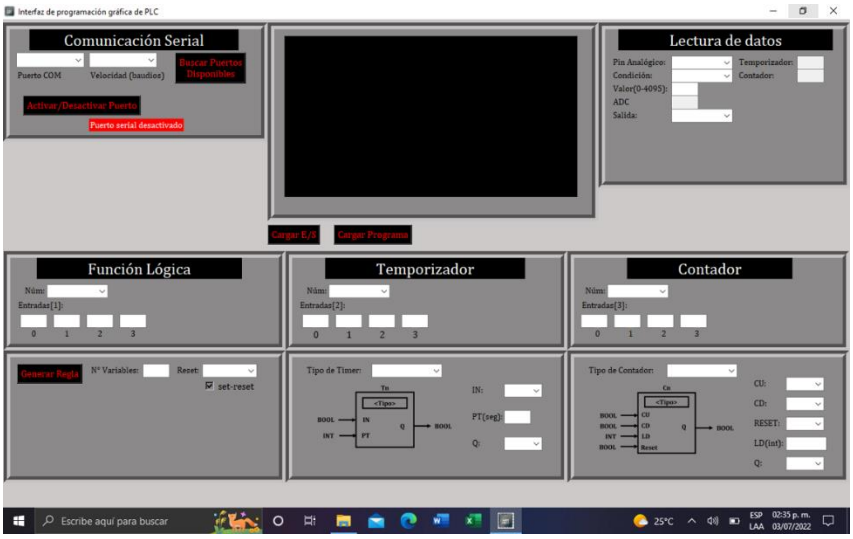

Figura 52. *Archivo EXE en una computadora de 32 bits marca Acer® sistema operativo Windows.*

En una computadora de 64 bits marca HP® sistema operativo Windows:

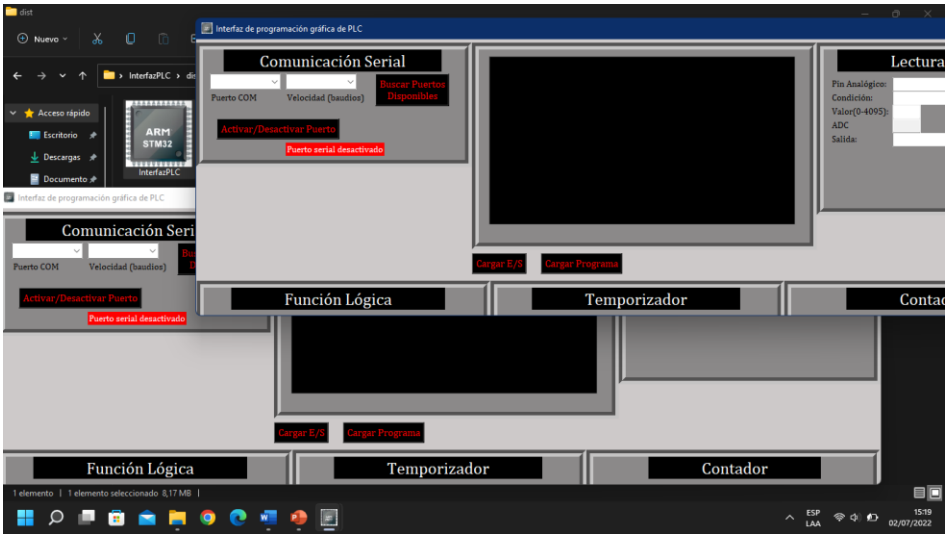

Figura 53. *Archivo EXE en una computadora de 64 bits marca HP® sistema operativo Windows.*

Conclusión.

Este proyecto tiene un alcance enorme si este presenta mejoras y/o actualizaciones continuas ya que puede llegar a sustituir el uso de los PLC en la automatización debido a su mayor ventaja, el precio de su uso.

El desarrollo de este proyecto es algo muy complejo si se quiere que un microcontrolador comercial emule las funciones de un PLC de una empresa, ya que está limitado por los conceptos del desarrollador del proyecto porque requiere conocimientos avanzados en la programación de interfaces gráficas, la programación orientada a objetos (POO) y conocer todas las funciones del lenguaje de programación que se quiera usar. Además el desarrollador debe visualizar una interfaz que sea intuitiva y permita una programación gráfica como en un PLC para que todos los usuarios que quieran usar la interfaz no tengan problemas a la hora de utilizarla.

Otro punto es que ser una interfaz diseñada por uno mismo, se pueden ofrecer distintas formas de programar un microcontrolador para hacerlo más fácil o entendible dándole distintos formatos que se pueden relacionar con otros campos de la electrónica. Como adecuar la programación Ladder de un PLC por un formato como una función lógica tradicional que es funcionalmente igual al esquema de contactos de un segmento del

PLC. Esto permitiría realizar instrucciones específicas para controlar otros dispositivos actuadores o de visualización y ampliar las aplicaciones de este proyecto.

Básicamente este proyecto logra su objetivo que es el controlar algunas funciones de un microcontrolador a través de una interfaz si necesidad de programarlo de manera común que hace uso de los elementos de programación avanzada para la automatización de procesos.

Mejoras a futuro.

Al ser un funcionamiento básico de un PLC este proyecto requerirá muchas mejoras que faciliten y eficiente la programación de diversos procesos de automatización y control.

La interfaz solo puede programar un solo elemento de cada herramienta de programación; solo una función, un temporizador y un contador por lo que la mejora sería el poder implementar n número de instrucciones de cada herramienta de programación.

Realizar ajustes en las excepciones de la interfaz para evitar que un operador realice un comando no apropiado que comprometa el funcionamiento de un sistema.

Implementar una mayor lectura de datos recibidos del puerto serial y visualizar más datos de sensores en distintos formatos como voltaje en tiempo real y gráfico.

Interconectar los elementos de programación mediante otra instrucción que no sea el uso de los pines de las salidas de otros elementos para que funcione como la programación en escalera o Ladder encontrada en los PLC’s.

Emplear otro sistema de visualización de la programación que se encuentra cargada en el microcontrolador, cambiar de pseudocódigo a algo más gráfico.

Un sistema de visualización del estado de cada entrada y salida del microcontrolador y así detectar a distancia una posible falla en una conexión física.

Hacer uso de elementos más avanzados como manipular sensores que requieran de otro tipo de comunicación como I2C y 1-Wire. Con ayuda de más sensores cabría la capacidad de crear un control PID y controlar un sistema de control en lazo cerrado.

Creación de módulos propios para el control de periféricos específicos de visualización y actuadores.

Referencias

Alves, T. (2022). *Open PLC*. Obtenido de FreeWave: https://www.openplcproject.com/

Badía, A. M. (1998). *Autómatas Programables.* Barcelona: MARCOMBO.

Breijo, E. G. (2008). *Compilador C CCS y Simulador Proteus para Microcontroladores PIC.* Barcelona: Alfaomega.

Controladores, P. P. (7 de Febrero de 2022). *CONTROLLINO, un PLC con software open source y compatible con Arduino*. Obtenido de infoPLC: https://www.infoplc.net/blogs-automatizacion/item/110810-controllino-plc-software-open-source-compatible-arduino

FESTO DIDACTIC. (2002). *Controlador Lógico Programable.* Obtenido de PDF DRIVE: https://es.pdfdrive.com/controlador-l%C3%B3gico-programable-e51843914.html

Industrial Shields. (2022). *Gama 20 Entradas-Salidas Arduino PLC*. Obtenido de Boot & Work Corp S.L.: https://www.industrialshields.com/es_ES/industrial-plc-based-on-arduino-original-boards-automation-solutions-20ios

Intesc Electronics & Embedded. (2017). *Tarjeta de desarrollo Ophyra.* Obtenido de Intesc: https://intesc.mx/productos/tarjeta-de-desarrollo-ophyra/

iottrends.tech. (2020). *How To Interface UART Communication Using MicroPython*. Obtenido de GADGETS: https://www.iottrends.tech/blog/how-to-interface-uart-communication-using-micropython/

Maocho, F. (6 de Febrero de 2016). *Arduino – Comunicaciones asíncronas, pines marcados como TX y RX*. Obtenido de Wordpress: https://felixmaocho.wordpress.com/2016/02/06/arduino-comunicaciones-asincronas-pines-marcados-como-tx-y-rx/

Pérez, E. M. (2009). *AUTÓMATAS PROGRAMABLES Y SISTEMAS DE AUTOMATIZACIÓN .* Barcelona: MARCOMBO.

Python Software Foundation. (25 de Abril de 2022). *tkinter — Interface de Python para Tcl/Tk*. Obtenido de Python Software Foundation License Version 2: https://docs.python.org/es/3/library/tkinter.html

Recursos Python. (22 de Abril de 2021). *Botón (Button) en Tcl/Tk (tkinter)*. Obtenido de Recursos Python: https://recursospython.com/guias-y-manuales/boton-button-en-tkinter/

Recursos Python. (25 de Febrero de 2022). *Caja de texto (Entry) en Tcl/Tk (tkinter)*. Obtenido de Recursos Python: https://recursospython.com/guias-y-manuales/caja-de-texto-entry-tkinter/

Recursos Python. (20 de Marzo de 2022). *Lista desplegable (Combobox) en Tcl/Tk (tkinter)*. Obtenido de Recursos Python: https://recursospython.com/guias-y-manuales/lista-desplegable-combobox-en-tkinter/

STMicroelectronics. (2022). *STM32F407VG*. Obtenido de ST life.augmented: https://www.st.com/en/microcontrollers-microprocessors/stm32f407vg.html

Tollervey, N. H. (2017). *Programming with MicroPython.* Sebastopol: O'Reilly books.

Anexos.

4.5. Comunicación serial entre Python y STM32 Micropython.

Antes de programar, el microcontrolador debe estar conectado directamente a la PC donde Python se encuentra instalado y ubicar que puerto COM le fue otorgado por la PC, regularmente con el nombre "USB Serial Port (COMX)", para ello se debe navegar hasta el panel de control>Administrador de dispositivos>Puertos (COM y LTP) de la PC.

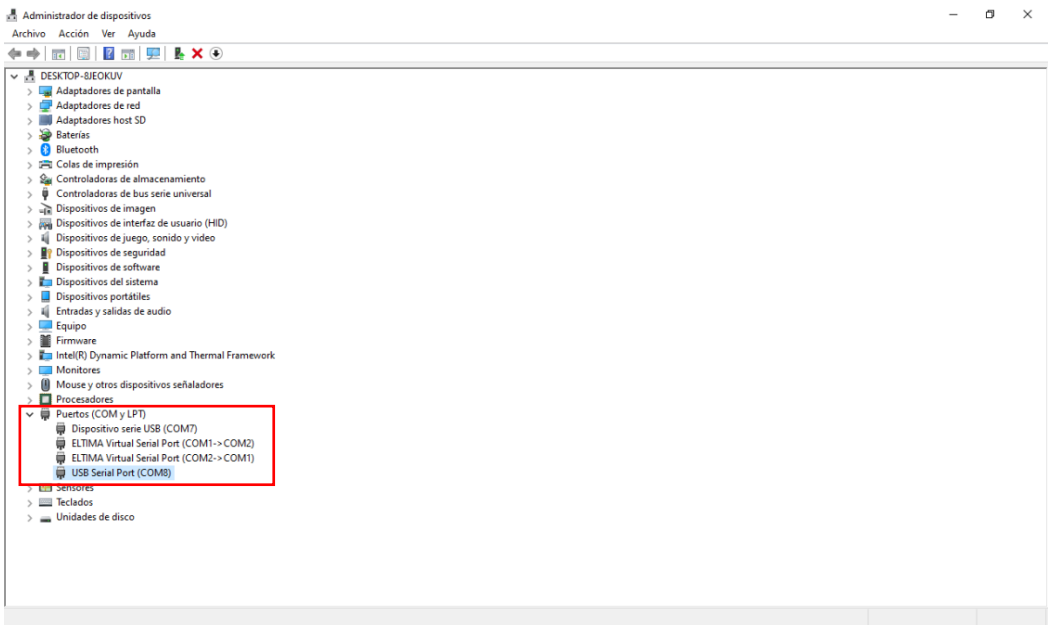

Figura 54. *Identificación de los puertos COM del microcontrolador.*

No se debe confundir con "Dispositivo serie USB (COMX)" ya que se trata de la conexión USB-OTG encargada de acceder a los archivos del sistema del microcontrolador STM32 y su respectiva programación.

Segundo, en Python se debe importar la librería para la comunicación serial:

```
import serial
```

Hacer un objeto de class `serial.Serial` y que a su vez contenga los parámetros necesarios para activar el puerto serial correctamente:

```
puerto = serial.Serial(port='COM8', baudrate=115200,
                       bytesize=serial.EIGHTBITS,
                       parity=serial.PARITY_NONE,
                       stopbits=serial.STOPBITS_ONE)
```

En este caso el puerto es COM8 y el microcontrolador STM32 comúnmente trabaja con una velocidad de 115200 baudios, el comando anterior ya hace la tarea de habilita el puerto COM por lo que solo falta una función o instrucción que se encargue de enviar los datos y de desactivar el puerto cuando ya no se necesite:

```
while 1:
   x = str(input('Dame un texto: ')) # Función bloqueante hasta que se detecte
un salto de línea.
   puerto.write(x.encode())   # Función que permite enviar los datos
codificados de manera binaria
   if x == 'exit':      # Ejemplo de condición para terminar el ciclo While
      break
puerto.close()                     # Cerrar el puerto COM
```

Tercero, en Micropython es importante identificar en que pines se encuentran TX y RX debido a que dependen del UART a elegir. En la tarjeta Ophyra de acuerdo a su manual la interfaz UART que utiliza es UART (3), pero antes debe importarse la librería UART de pyb:

```
from pyb import UART
```

El UART debe iniciarse con algunos parámetros principalmente con la misma velocidad que se maneja en Python:

```
uart = UART(3, 115200, bits=8, parity=None, stop=1,timeout=0)

uart.init(115200)
```

Solo falta la función principal o ciclo para enviar y/o recibir los datos (los datos son recibidos están codificados en bits, por ejemplo, `b'palabra'`):

```
while True:
    ch = uart.read()  # Lee el dato en el puerto COM
    if ch:            # La condición no se ejecuta hasta recibir un dato
       print(ch)
       ch = b''       #  Borra  la  variable  para  no  volver  a  ejecutar  la
condición
    if ch == b'exit': # Finaliza el ciclo
       break
```

4.6. Declaración de algunos Widgets en interfaz TKinter

Botón (Button) en Tcl/Tk (tkinter)

Un botón es un recuadro con un texto y/o una imagen que puede ser presionado por el usuario para ejecutar una operación. En Tk está representado por las clases tk.Button y ttk.Button. (Recursos Python, 2021)

Para obtener un botón se debe crear una instancia de la clase correspondiente:

```
import tkinter as tk
from tkinter import ttk

root = tk.Tk()
root.config(width=300, height=200)
root.title("Botón en Tk")

boton = ttk.Button(text="¡Hola, mundo!")
boton.place(x=50, y=50)
   root.mainloop()
```

Para darle funcionalidad a un botón, se debe crear una función y luego asignársela con ayuda del argumento command. Por ejemplo, el siguiente código imprime ¡Hola, mundo! en la consola cuando el usuario presiona el botón:

```
import tkinter as tk
```

```
from tkinter import ttk

def saludar():
    print("¡Hola, mundo!")

root = tk.Tk()
root.config(width=300, height=200)
root.title("Botón en Tk")

  boton = ttk.Button(text="¡Hola, mundo!", command=saludar)
  boton.place(x=50, y=50)

  root.mainloop()
```

Caja de texto (Entry) en Tcl/Tk (tkinter)

Permite al usuario ingresar cualquier texto de una línea. En Tcl/Tk está representada a través de la clase `ttk.Entry`, que a su vez hereda la funcionalidad de un control más primitivo llamado `tk.Entry`.

Para crear una caja de texto, se crea una instancia de la primera clase:

```
import tkinter as tk
from tkinter import ttk

root = tk.Tk()
root.config(width=300, height=200)
# Crear caja de texto.
entry = ttk.Entry()
# Posicionarla en la ventana.
entry.place(x=50, y=50)
  root.mainloop()
```

Operaciones con Entry.

Método	Descripción	Ejemplo

`get()`	Para obtener lo que el usuario ha escrito en la caja de texto.	`entry.insert(0, "Hola mundo!")`
		`entry.place(x=50, y=50)`
		`button = ttk.Button(text="Obtener texto", command=lambda: print(entry.get()))`
		`button.place(x=50, y=100)`

`insert()`	Para añadir un texto se emplea que toma como primer argumento una posición y como segundo una cadena.	`entry.insert(0, "¡Hola,")` `entry.insert(6, " mundo!")`

Tabla 5. *Ejemplos de algunas operaciones contenidas en la clase Entry.*

Asociar una variable.

Tk provee la clase `tk.StringVar()` para crear objetos que actúan como una cadena, con la excepción de que para asignarle un valor se usa el método `set()` y, para obtenerlo, `get()`.

Se puede asociar una variable de estas características a una caja de texto al momento de su creación, a través del parámetro `textvariable`. (Recursos Python, 2022)

```
1. entry_var = tk.StringVar()
2. entry = ttk.Entry(textvariable=entry_var)
```

Lista desplegable (Combobox) en Tcl/Tk (tkinter)

Se trata de una combinación entre una lista y una caja de texto. Así, puede actuar como una lista desplegable con determinadas opciones y eventualmente permitir al usuario escribir un valor que no se encuentre en la lista.

Para añadir un conjunto de opciones, pasamos una lista al argumento values.

```
combo = ttk.Combobox(
```

```
    state="readonly",
    values=["Python", "C", "C++", "Java"]
)
```

Un combobox en Tk trabaja únicamente con cadenas. Cualquier otro tipo de objeto será convertido antes de ser añadido a la lista.

El método `combobox.get()` retorna la opción seleccionada: ya haya sido esta seleccionada a partir de una de las opciones, ya haya sido escrita manualmente por el usuario. (Recursos Python, 2022)

```
1. from tkinter import messagebox, ttk
2. import tkinter as tk
3.
4. def show_selection():
5.     # Obtener la opción seleccionada.
6.     selection = combo.get()
7.     messagebox.showinfo(
8.     message=f"La opción seleccionada es: {selection}",
9.     title="Selección"
10.   )
11.
12.   main_window = tk.Tk()
13.   main_window.config(width=300, height=200)
14.   main_window.title("Combobox")
15.   combo = ttk.Combobox(
16.       state="readonly",
17.       values=["Python", "C", "C++", "Java"]
18.   )
19.   combo.place(x=50, y=50)
20.   button = ttk.Button(text="Mostrar selección", command=show_selection)
21.   button.place(x=50, y=100)
22.   main_window.mainloop()
```

El método get() siempre retorna una cadena y, en caso de no haber ninguna selección, devuelve una cadena vacía.

También es posible obtener el índice (es decir, su posición en la lista comenzando desde el 0) del elemento seleccionado vía current().

```
index = combo.current()
```

4.7. Convertir un archivo py a exe.

Python cuenta con una librería llamada “pyinstaller”, que convierte un archivo de Python a uno ejecutable y pueda ser usado en cualquier computadora de que acepte este tipo de archivo sin necesidad de correr el programa en un editor de Python y no estar instalando las librerías necesarias.

En primer lugar dicha librería debe estar instalada de manera global en la PC, esto se logra mediante un comando en CMD que al correrlo comenzará con la instalación de la librería, el comando es:

```
pip install pyinstaller
```

Segundo, se recomienda crear una carpeta de X nombre en que se creará el archivo y que sea de fácil acceso, por ejemplo en el escritorio de la PC:

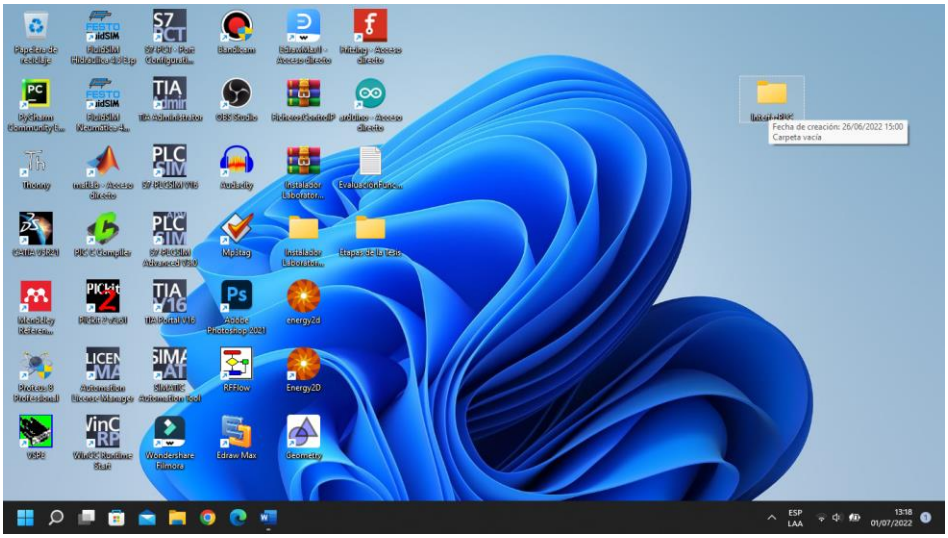

Figura 55. *Creación de la carpeta InterfazPLC.*

Tercero, copiar el archivo py que se desea convertir a ejecutable; el software PyCharm los crea en la dirección (en este caso se trata de una computadora HP):

"C:\Users\HP\PycharmProjects\InterfazPLC\NombreArchivo.py", el copiar el archivo facilita su conversión ya que al convertirlo se crean diversos archivos y con la carpeta creada solo estará disponible el archivo exe y su respectivo archivo py.

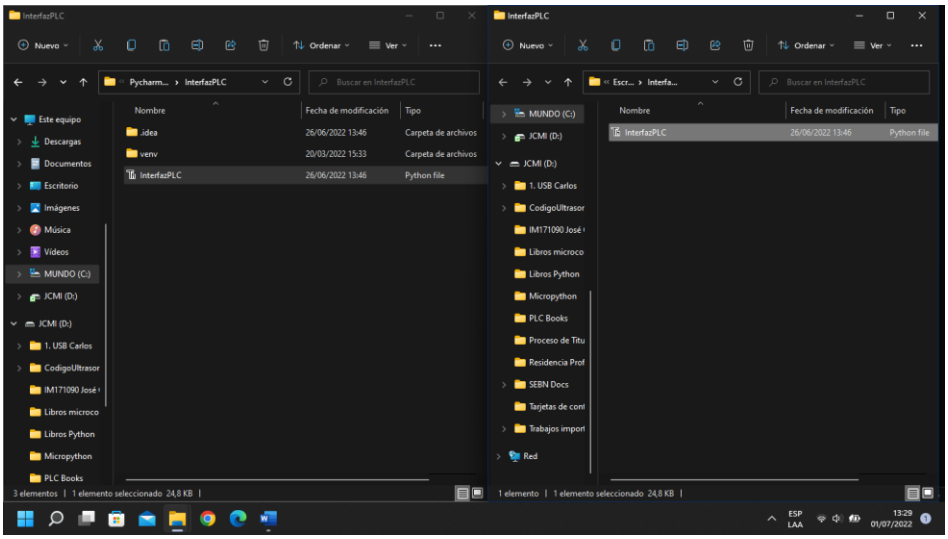

Figura 56. *Copia del archivo py del proyecto a la carpeta recién creada.*

Antes de pasar al siguiente paso, la terminal de CMD debe abrirse en PyCharm ya que se requiere de un **entorno virtual**.

Otro punto importante es si la interfaz usa imágenes que se encuentran en el equipo en el que se creó dicha interfaz, también deben estar en la carpeta en donde se va a crear el archivo ejecutable, esta interfaz usa dos imágenes .png y una .ico, que obligatoriamente deben estar junto al archivo de Python para ser declaradas solo con su nombre y extensión de archivo ya que si tienen toda la dirección el archivo ejecutable no será capaz de encontrar las imágenes a la hora de abrir el archivo.

Correcto	`image = r'imagen.png'`
Incorrecto	`image = r'C:Users/HP/desktop/imagen.png'`

Tabla 6. *Diferencia entre la imagen cuando esta en el mismo directorio y cuando está en otro directorio.*

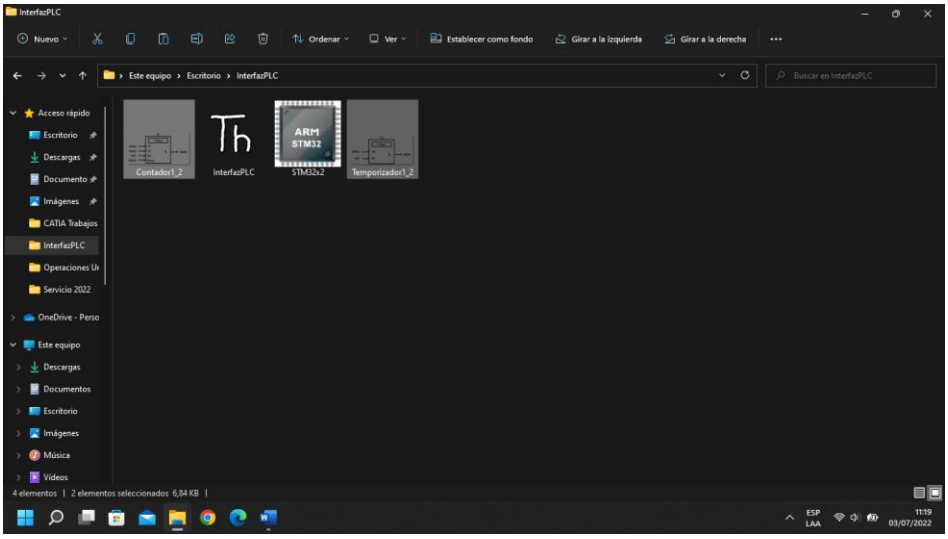

Figura 57. *Archivo de Python junto a sus componentes o imágenes.*

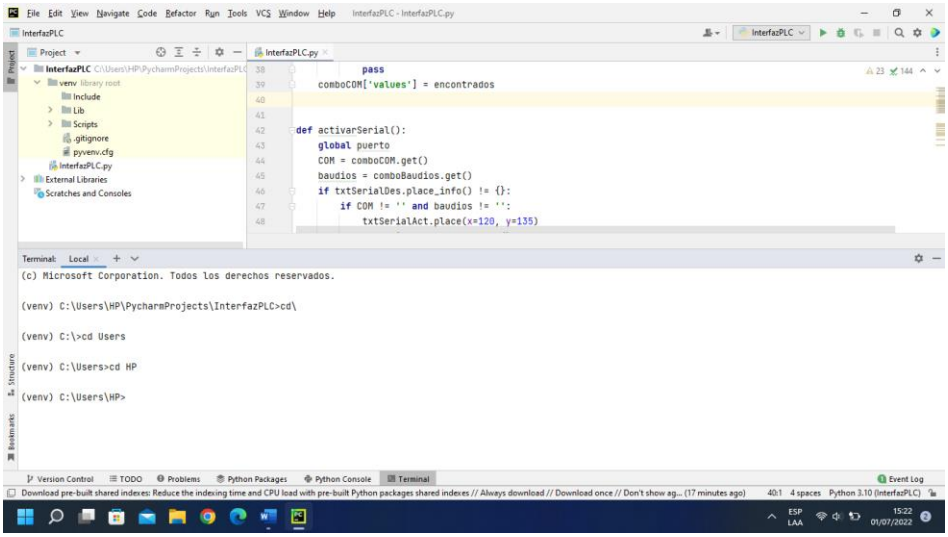

Figura 58. *CMD desde PyCharm.*

Cuarto, hacer uso del comando cd para ingresar a las subcarpetas de una ventana; en este caso el archivo py esta en la carpeta InterfazPLC que está ubicada en el escritorio por lo que el comando en CMD será:

```
cd "C:\Users\HP\OneDrive\Escritorio\InterfazPLC"
```

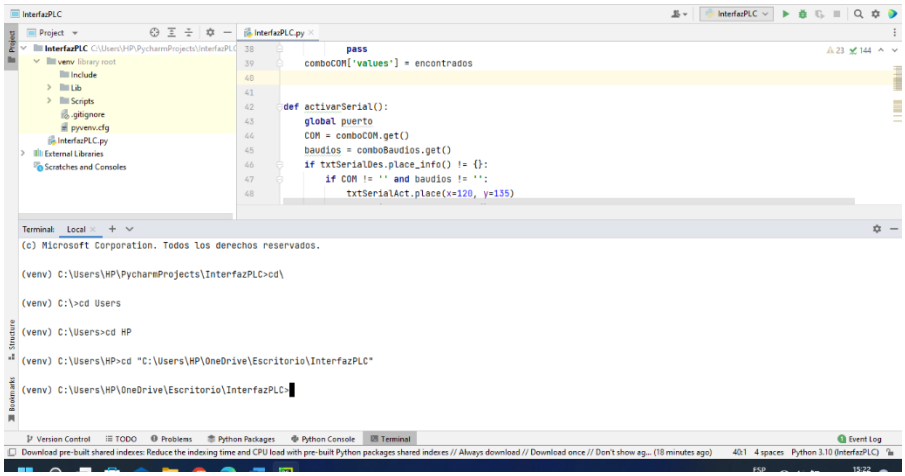

Figura 59. *Comando introducido parta comenzar con el proceso de conversión.*

Una vez que se ha accedido a la carpeta se tienen tres comandos al convertir un exe:

Al correr el exe, si no se retira el fondo de CMD cada que se ejecuta un archivo exe se va a quedar estática una ventana que no tiene ninguna función y da una mala imagen de un archivo por lo que el comando para eliminar esa pantalla es `pyinstaller --windowed archivo.py`

Otro caso es que la creación genera muchos ficheros que no sirven de nada y solo ocupan espacio por lo que el comando para borrarlos es `pyinstaller --onefile archivo.py`

Y se puede cambiar el icono, este es de manera opcional el cual requiere de una imagen en formato .ico que este en la misma carpeta o directorio en el que se encuentra el script de Python con el comando `pyinstaller --icon=./icono.ico archivo.py`

Si se quieren que todos los comandos formen parte del archivo ejecutable, el comando completo es:

`pyinstaller --windowed--onefile--icon=./icono.ico archivo.py`

Donde el `archivo.py` es el nombre del archivo a convertir.

En este caso el archivo se llama InterfazPLC y el icono tiene por nombre STM32x2 por lo que solo falta sustituir:

```
pyinstaller --windowed --onefile --icon=./STM32x2.ico InterfazPLC.py
```

Solo falta correr el comando y esperar. Al verificar la carpeta se tendrán nuevos archivos, el importante es la carpeta “dist” que es donde está el ejecutable, las imágenes que contiene la interfaz deberán ser copiadas a esa carpeta “dist” y estará listo para usar y compartir.

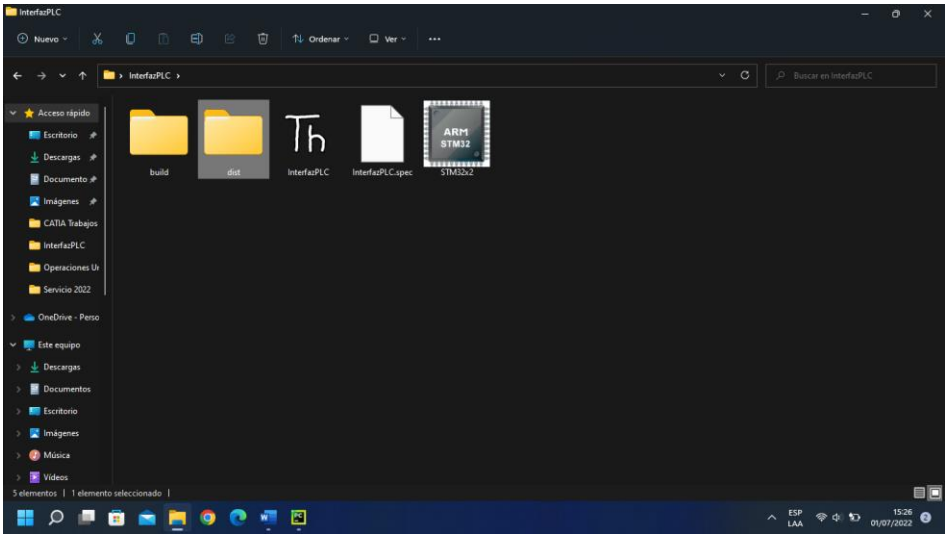

Figura 60. *Nuevos archivos generados por el comando pyinstaller.*

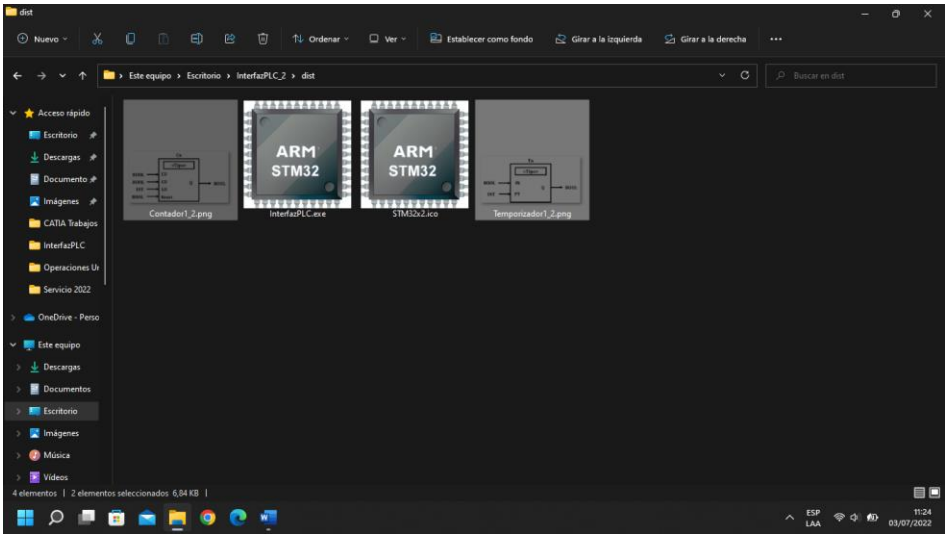

Figura 61. *Archivo ejecutable junto a las imágenes.*

Al abrirlo se tendrá la interfaz.

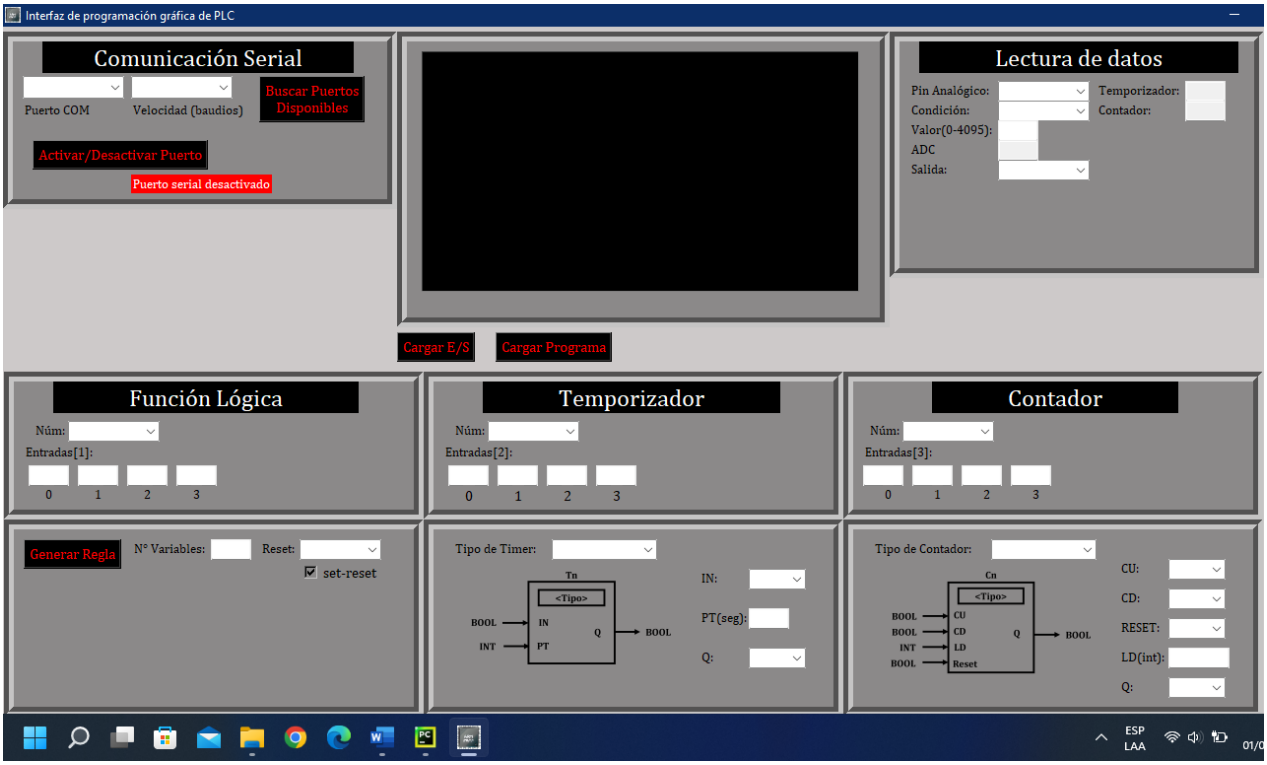

Figura 62. *Interfaz del archivo ejecutable.*