ANEXO I

MANUAL DE USUARIO

En este manual se explica cómo usar de forma correcta el sistema SCADA del bloque de acondicionamiento de la planta de ensayos y operación de reformado de metano.

Inicialización del sistema

Encender el Intel Galileo conectando el transformador de 2 A a 5 V.

Encender los controladores de flujo conectando los transformadores de 1 A a 24 V.

Conectar los controladores de flujo al Intel Galileo.

Conectar el Intel Galileo al puerto Ethernet de la computadora.

Abrir la aplicación de escritorio de la interfaz hombre-máquina, al realizar esto se mostrará la ventana que se observa en la figura 1.

Figura 1. Ventana de inicio.

Ingreso al sistema SCADA

Para ingresar al sistema se requiere colocar un nombre de usuario y contraseña, y presionar el botón de “INICIAR SESIÓN”, tal y como se observa en la figura 2.

Figura 2. Inicio de sesión

Al ingresar de forma correcta el nombre de usuario y la contraseña se muestra un mensaje que indica que el inicio de sesión ha sido exitoso y cual usuario ha iniciado sesión. En caso contrario el sistema mostrará un mensaje indicado el error, tal y como se observa en la figura 3.

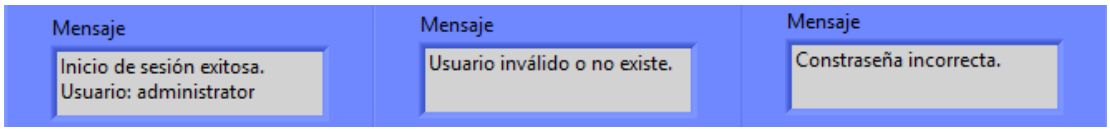

Figura 3. Mensajes de inicio de sesión.

Uso de la interfaz humano máquina

Luego de iniciar sesión correctamente se habilitarán las pestañas de la aplicación, la primera es la de "ADMINISTRADOR DE USUARIO", que corresponde a la ventana de inicio y desde la cual se puede cerrar sesión o apagar el equipo, como se muestra en la figura 4.

Figura 4. Pestaña de administrador de usuario.

Al cerrar sesión el sistema mostrará un mensaje indicado que se ha cerrado sesión correctamente, ver la figura 5, y al seleccionar el botón de "APAGAR INTEL GALILEO" aparecerá una ventana emergente para confirmar la selección, al apretar "Sí" el sistema mostrara un mensaje por dos segundos y cerrará la aplicación, en la figura 6 se muestran la ventana emergente y el mensaje de apagar el equipo.

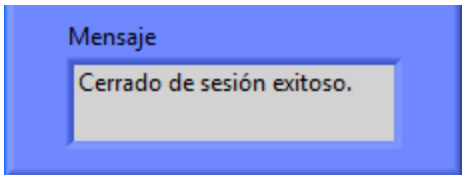

Figura 5. Mensaje de cerrado de sesión.

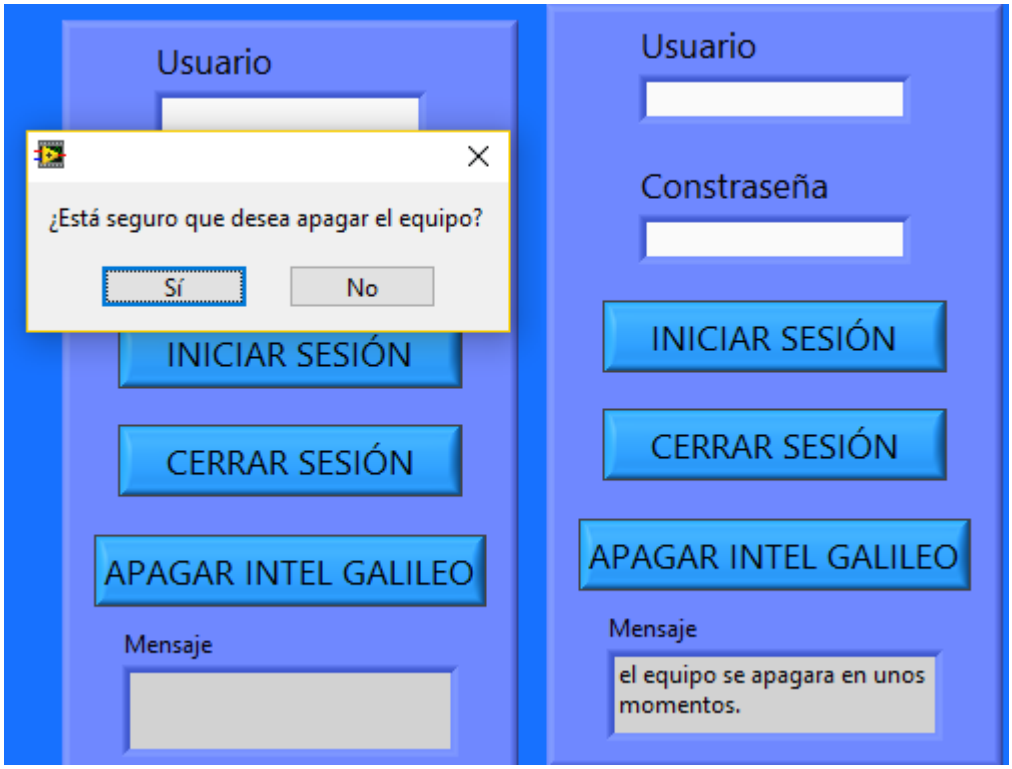

Figura 6. Ventana emergente y mensaje para apagar el equipo.

La segunda pestaña se llama "INFORMACIÓN DE LOS CONTROLADORES", en la cual se muestra el estado de las variables en tiempo real y se pueden ejecutar las acciones de control supervisorio, en la figura 7 se observa esta pestaña.

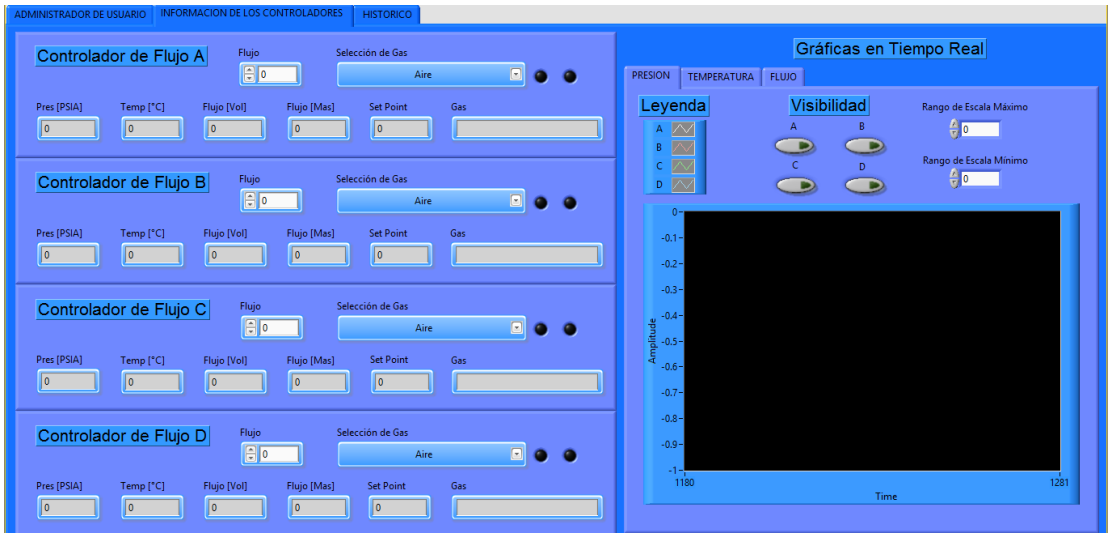

Figura 7. Pestaña de información de los controladores.

Para realizar acciones de control supervisorio se disponen por cada dispositivo de un control numérico que fija el punto de consigna y un control de selección del gas a monitorear, en la figura 8 se observan estos elementos.

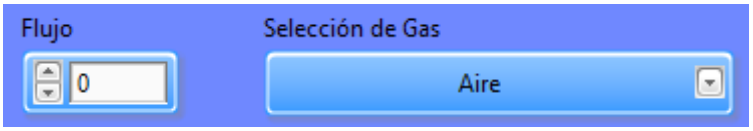

Figura 8. Elementos para realizar acciones de control supervisorio.

El punto de consigna está limitado a valores entre 0 y 1 SLPM, con una precisión de 0.001 SLPM; y la selección de gas muestra una lista de los 30 gases disponibles en los controladores para escoger.

El monitoreo de las variables se realiza a través de los indicadores numéricos que muestran los valores de presión, temperatura, flujo volumétrico, flujo másico, y punto de consigna y el indicador de texto que muestra el aire controlado, tal y como se observa en la figura 9.

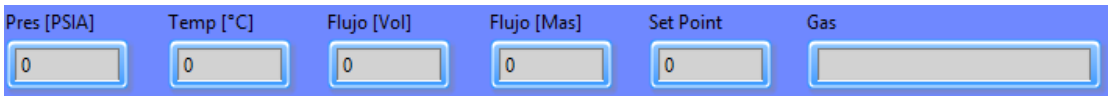

Figura 9. Elementos para monitorear el estado de los gases.

Además se pueden visualizar en tres gráficas los parámetros de presión, temperatura y flujo volumétrico del gas, que muestran el valor en tiempo real de la variable, para cambiar entre los gráficos se disponen de tres pestañas, nombradas según la variable que muestran. Estas gráficas cuentan cada una con cuatro botones de visibilidad para ocultar o mostrar alguna curva y dos controles numéricos para controlar la escala de la gráfica, en la figura 10 se muestra esta sección.

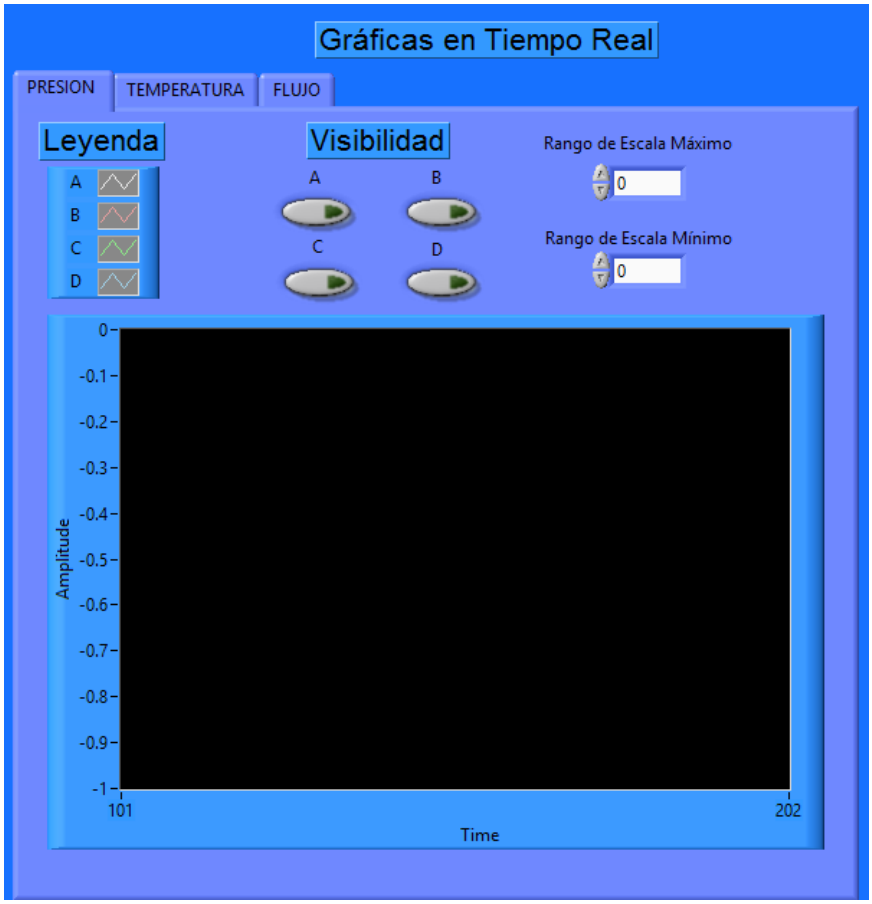

Figura 10. Gráficas en tiempo real de las variables.

La tercera pestaña tiene como nombre “HISTÓRICO”, en ella se muestra el registro de datos automático a través de tres gráficas, en las cuales se muestran los valores de las variables de mayor interés según la fecha. Al igual que antes cada una cuentan con cuatro botones de visibilidad y dos controles numéricos para modificar la escala, además incluye una barra de herramientas para interactuar con la gráfica, ver la figura 11.

Figura 11. Pestaña de histórico.

El sistema cuenta con dos alarmas en forma de LED que se activan y detienen el sistema al ocurrir alguna falla, se desactivan cuando la falla es solucionada, en la figura 12 se observan los LED encendidos.

Figura 12. LED que señalan la activación de las alarmas

ANEXO II
INTRUCTIVO DE CÓDIGO FUENTE DEL INTEL GALILEO

Este instructivo tiene como objetivo exponer las principales funciones utilizadas en el desarrollo del programa diseñado en el sistema embebido Intel Galileo para la adquisición y procesamiento de datos provenientes de los controladores de flujo. Antes de explicar las funciones se presentará las primeras líneas de código, en donde se importan las librerías, se declaran las variables y se realizan las configuraciones iniciales.

Script principal

```
#!/usr/bin/python

#Importar el modulo de tiempo
import time

#Importar la clase serial desde la libreria pyserial
import serial

#Importar la clase de GPIOGalileo desde el modulo wiringx86
from wiringx86 import GPIOGalileo as GPIO

#Importar el modulo de expresiones regulares
import re

#Importar modulo de estructuras
import struct

#Importar la clase de CLiente ModbusTcp
from pymodbus.client.sync import ModbusTcpClient

#Importar módulo para ejecutar instrucciones en la shell de Linux
import commands
```

Figura 1. Importación de librerías.

En la figura 1 se muestran las primeras líneas del código en las cuales se importan todas las librerías a utilizar. Es de vital importancia la primera línea de

código en la cual se específica el directorio en que se tiene instalado Python, esto es con la finalidad de que el sistema identifique con cual interprete debe leer el script.

```
#Creando nuevo objeto Cliente ModbusTcp
cliente = ModbusTcpClient('169.254.9.102',502)

#Creando un nuevo objeto GPIO
#Si la opcion de debug se configura como True muestra la interraccion con sysfs
gpio = GPIO(debug = False)

#Creando nuevo objeto serial
ser = serial.Serial()
```

Figura 2. Declaración de objetos y clases.

En la figura 2 se observa la declaración de los objetos, a través de los cuales se llaman las funciones de las librerías utilizadas para la comunicación serial, el protocolo Modbus y manejar los GPIO del Intel Galileo.

```
#Definiendo lista de aires
AIRES=['Air','Ar','CH4','CO','CO2','C2H6','H2','He','N2','N2O','Ne','O2','C3H8','n-C4H10',
'C2H2','C2H4','i-C4H10','Krypton','Xe','SF6','C-25','C-10','C-8','C-2','C-75','A-75','A-25','A1025','Star29','P-5']

#Definiendo lista de controladores
CONTROLADORES = ['A','B','C','D']

#Definiendo variables
selA = 2
selB = 3
inha = 4

#Definir variables para guardar los datos anteriores
pointaA = 0
pointaB = 0
pointaC = 0
pointaD = 0
airaA = 0
airaB = 0
airaC = 0
airaD = 0
```

Figura 3. Definición de variables y tablas.

La figura 3 muestra la definición de las tablas usadas para detectar el gas y realizar el multiplexado, y las variables empleadas para manipular los GPIO y comparar el punto de consigna y el aire que se manda a escribir a través de las acciones de control supervisorio.

```
#Configurando pins como salidas
print 'Configurando pins'
gpio.pinMode(selA, gpio.OUTPUT)
gpio.pinMode(selB, gpio.OUTPUT)
gpio.pinMode(inha, gpio.OUTPUT)

#Configurando caracteristicas de la comunicacion
print 'Configurando la comunicacion'
ser.port = '/dev/ttyS0'
ser.baudrate = 19200
ser.bytesize = serial.EIGHTBITS
ser.parity = serial.PARITY_NONE
ser.stopbits = serial.STOPBITS_ONE
ser.timeout = 0.06
ser.xonxoff = False
ser.rtscts = False
ser.dsrdtr = False
ser.write_timeout = None
ser.inter_byte_timeout = None

#Habilitando comunicacion
print 'Abirendo el canal de comunicacion'
ser.open()

#Conectando cliente Modbus al Servidor
print 'Conectando al Servidor ModbusTcp'
cliente.connect()
```

Figura 4. Configuración inicial y conexión de las comunicaciones.

En la figura 4 se observa la configuración de los GPIO utilizados del Intel Galileo y la comunicación serial, además se abre el canal de comunicación y se conecta el cliente Modbus al servidor.

```
#Definir funcion para convertir float a entero sin signo


def binario(num):
    packed = struct.pack('!f',num)
    integers = [ord(c) for c in packed]
    binaries = [bin(i) for i in integers]
    stripped = [s.replace('0b', '') for s in binaries]
    padded = [s.rjust(8, '0') for s in stripped]
    HB = int(''.join([padded[0],padded[1]]), 2)
    LB = int(''.join([padded[2],padded[3]]), 2)
    bit = [HB,LB]
    return bit


#-------------------------------------------------------------
```

Figura 5. Función para convertir de flotante a entero sin signo.

En la figura 5 se observa la función que se encarga de convertir de flotante a entero sin signo. En principio se utiliza el comando pack de la librerías struct para separar el número flotante en cuatro caracteres, que son representaciones de los 4 bytes que conforman el número; luego se transforman dichos caracteres en enteros, que van desde 0 a 255; inmediatamente se expresan de forma binaria, quedando con el formato “0b10001011”; los dos comandos siguientes se utilizan para eliminar los caracteres “0b” y completar con 0 hasta llegar a 8 caracteres aquellos valores que podían ser representado con menos dígitos, tal como un 7, que se representaría “0b111”; en los dos comandos sucesivos se juntan los bits en dos grupos de 16, donde cada uno representa un entero sin signo y se guardan los datos en un vector; para finalizar se devuelve el valor del vector.

```
#------------------------------------------------------------
#Definiendo funcion de comprobacion de trama


def ComprobacionTrama(Datos, Lista=AIRES):
    contador = 0
    if re.search('^[A-Z] \+\d{3}\.\d{2} \+\d{3}\.\d{2} \+\d{2}\.\d{3} \+\d{2}\.\d{3} \d{2}\.\d{3}\s+\S+', Datos):
        contr = Datos[0]
        press = binario(float(Datos[2:9]))
        tempe = binario(float(Datos[10:17]))
        volf = binario(float(Datos[18:25]))
        massf = binario(float(Datos[26:33]))
        setp = binario(float(Datos[34:40]))
        airl = re.findall('^[A-Z] \+\d{3}\.\d{2} \+\d{3}\.\d{2} \+\d{2}\.\d{3} \+\d{2}\.\d{3} \d{2}\.\d{3}\s+(\S+)', Datos)
        air = airl[0]
        for x in Lista:
            contador = contador + 1
            if air == x:
                if contr == 'A':
                    direccion = 0
                if contr == 'B':
                    direccion = 11
                if contr == 'C':
                    direccion = 22
                if contr == 'D':
                    direccion = 33
                cliente.write_register(direccion+0, press[0])
                cliente.write_register(direccion+1, press[1])
                cliente.write_register(direccion+2, tempe[0])
                cliente.write_register(direccion+3, tempe[1])
                cliente.write_register(direccion+4, volf[0])
                cliente.write_register(direccion+5, volf[1])
                cliente.write_register(direccion+6, massf[0])
                cliente.write_register(direccion+7, massf[1])
                cliente.write_register(direccion+8, setp[0])
                cliente.write_register(direccion+9, setp[1])
                cliente.write_register(direccion+10, contador)


#------------------------------------------------------------------------
```

Figura 6. Funcion para comprobar la calidad de la trama.

En la figura 6 se observa la función para comprobar si la trama enviada por los controladores de flujo contiene errores. Se declara un contador que se utiliza para comparar el gas recibido con los registrados en la tabla. Primero se utiliza la librería de expresiones regulares para crear una expresión con el formato de una trama sin errores, la cual se utiliza para compararla con los datos recibidos, si el resultado es afirmativo se extraen los valores de cada parámetro, los cuales se convierten de cadena de caracteres a flotante y luego a enteros sin signo, además se extrae la cadena de caracteres que representa el gas del final de la trama y se compara con la tabla declarada en el inicio, si nuevamente el resultado es acertado se comprueba que la trama no contiene errores, por lo que finalmente los valores de los parámetros se escriben en los bloques de memoria del protocolo Modbus.

```
def Escribir(controlador, pointa, aira):
    gpio.digitalWrite(inha, gpio.LOW) #Habilitacion multiplex
    #Seleccion de canal B = 0, A = 0
    if controlador == 'A':
        gpio.digitalWrite(selA, gpio.LOW)
        gpio.digitalWrite(selB, gpio.LOW)
        direc = 0
    #Seleccion de canal B = 0, A = 1
    elif controlador == 'B':
        gpio.digitalWrite(selA, gpio.HIGH)
        gpio.digitalWrite(selB, gpio.LOW)
        direc = 2
    #Seleccion de canal B = 1, A = 0
    elif controlador == 'C':
        gpio.digitalWrite(selA, gpio.LOW)
        gpio.digitalWrite(selB, gpio.HIGH)
        direc = 4
    #Seleccion de canal B = 1, A = 1
    elif controlador == 'D':
        gpio.digitalWrite(selA, gpio.HIGH)
        gpio.digitalWrite(selB, gpio.HIGH)
        direc = 6
    else:
        gpio.digitalWrite(inha,gpio.HIGH) #Deshabilitacion multiplex
    point = cliente.read_holding_registers(direc+44,1).getRegister(0)
    aire = cliente.read_holding_registers(direc+45,1).getRegister(0)
    if pointa != point:
        ser.write(controlador)
        time.sleep(0.0001)
        ser.write(str(point))
        time.sleep(0.0001)
        ser.write(b'\r')
        time.sleep(0.0001)
        pointa = point
    if aira != aire:
        ser.write(controlador)
        time.sleep(0.0001)
        ser.write(b'$')
        time.sleep(0.0001)
        ser.write(b'$')
        time.sleep(0.0001)
        ser.write(str(aire))
        time.sleep(0.0001)
        ser.write(b'\r')
        time.sleep(0.0001)
        aira = aire
    var = [pointa, aire]
    gpio.digitalWrite(inha, gpio.HIGH) #Deshabilitancion multiplex
    return var
```

Figura 7. Función que lleva a cabo las acciones de control.

En la figura 7 se muestra la función que se encarga de llevar a cabo las acciones de control supervisorio. Primero dependiendo de cuál controlador se necesite ajustar es seleccionado un canal del multiplexor y una dirección de memoria para leer los datos de control, luego se comparan los datos leídos con los que se escribieron en el ciclo anterior y si son diferentes se escribe el nuevo valor, en caso contrario la función no ejecuta ninguna acción. Esto se realiza para evitar enviar información innecesaria a través del canal de comunicación, ya que no se está efectuando ningún cambio, al final se guarda el valor del punto de consigna y del aire que se leyó en este ciclo para usarlo en el siguiente.

```
def Adquisicion(controlador):
    gpio.digitalWrite(inha, gpio.LOW) #Habilitacion multiplex
    #Seleccion de Canal B = 0, A = 0
    if controlador == 'A':
        gpio.digitalWrite(selA, gpio.LOW)
        gpio.digitalWrite(selB, gpio.LOW)
    #Seleccion de Canal B = 0, A = 1
    elif controlador == 'B':
        gpio.digitalWrite(selA, gpio.HIGH)
        gpio.digitalWrite(selB, gpio.LOW)
    #Seleccion de Canal B = 1, A = 0
    elif controlador == 'C':
        gpio.digitalWrite(selA, gpio.LOW)
        gpio.digitalWrite(selB, gpio.HIGH)
    #Seleccion de Canal B = 1, A = 1
    elif controlador == 'D':
        gpio.digitalWrite(selA, gpio.HIGH)
        gpio.digitalWrite(selB, gpio.HIGH)
    else:
        gpio.digitalWrite(inha, gpio.HIGH) #Deshabilitacion multiplex
    ser.write(controlador)
    time.sleep(0.0001)
    ser.write(b'\r')
    data = ser.read(50)
    ComprobacionTrama(data)
    gpio.digitalWrite(inha, gpio.HIGH) #Deshabilitiacion multiplex
```

Figura 8. Función que adquiere los datos de los controladores de flujo.

En la figura 8 se observa la función que se encarga de adquirir los datos de los controladores. Al igual que en la función anterior primero se selecciona el canal del multiplexor, luego se realiza la petición, se adquieren los datos y se comprueba la calidad de la trama, en donde como se mostró anteriormente, se escribe la información en los registros Modbus.

```
print 'Iniciando ciclo de Scan'


while (True):
    apagar = cliente.read_coils(0,1).getBit(0)
    if apagar == 1:
        cliente.write_coil(0,0)
        break
    else:
        for x in CONTROLADORES:
            Adquisicion(x)
            if x == 'A':
                var = Escribir(x, pointaA, airaA)
                pointaA = var[0]
                airaA = var[1]
            elif x == 'B':
                var = Escribir(x, pointaB, airaB)
                pointaB = var[0]
                airaB = var[1]
            elif x == 'C':
                var = Escribir(x, pointaC, airaC)
                pointaC = var[0]
                airaC = var[1]
            elif x == 'D':
                var = Escribir(x, pointaD, airaD)
                pointaD = var[0]
                airaD = var[1]
print '\nCerrando los canales de comunicación'
cliente.close()
ser.close()
print '\nLimpiando los pines'
gpio.digitalWrite(selA, gpio.LOW)
gpio.digitalWrite(selB, gpio.LOW)
gpio.digitalWrite(inha, gpio.HIGH)
print 'Apagando equipo'
commands.getoutput('shutdown -h')
```

Figura 9. Clico de escaneo.

En la figura 9 se llaman las funciones principales de adquisición de datos y de control, además se comprueba continuamente el estado de la bandera que indica cuando se debe apagar el equipo. En caso de que la bandera se active se sale del ciclo infinito, se cierran los canales de comunicación, se inhabilita el multiplexor y se ejecuta un comando en la Shell de Linux para apagar el dispositivo.

Script del servidor Modbus

```
#!/usr/bin/python

#Importar la implementacion del servidor
from pymodbus.server.sync import StartTcpServer

#Importar la clase para identificar el servidor
from pymodbus.device import ModbusDeviceIdentification

#Importar las clases para definir los bloques de datos
from pymodbus.datastore import ModbusSequentialDataBlock, ModbusSlaveContext, ModbusServerContext

#Inicializacion de la informacion del servidor
identidad = ModbusDeviceIdentification()
identidad.VendorName = 'Carlos Pagliarone'
identidad.ProductCode = 'Prototype'
identidad.ProductName = 'Server Modbus'
identidad.ModelName = 'Server Modbus Galileo'
identidad.MajorMinorRevision = '1.0'
print 'Datos del Servidor Inicializados'

#Inicializar el Almacenamiento de Datos
almacen = ModbusSlaveContext(
    di = ModbusSequentialDataBlock(0, [0]*100),
    co = ModbusSequentialDataBlock(0, [0]*100),
    hr = ModbusSequentialDataBlock(0, [0]*100),
    ir = ModbusSequentialDataBlock(0, [0]*100))
contexto = ModbusServerContext(slaves=almacen, single=True)
print 'Bloque de Datos de Almacenamiento Definidos'

#Iniciar el servidor
print 'Iniciando Servidor'
StartTcpServer(contexto, identity=identidad,address=("169.254.9.102",502))
```

Figura 10. Script del servidor Modbus.

En la figura 10 se tiene, al igual que en el principal, la sección donde se importan las librerías y se declara el directorio donde está instalado Python. Este script realiza la configuración del servidor Modbus, define los bloques de memoria, la dirección y puerto a utilizar para conectarse a él, finalmente inicia el proceso del servidor.

Script de arranque

```
#!/bin/sh
# /etc/init.d/confIP.sh

#Script para la configuracion inicial de la ip

### BEGIN INIT INFO
# Provides:             confIP.sh
# Required-Start:       $remote_fs $syslog
# Required-Stop:        $remote_fs $syslog
# Default-Start:        2 3 4 5
# Default-Stop:         0 1 6
# Short-Description:Script para la configuracion IP
# Description:          Configura la IP estatica en 169.254.9.102
### END INIT INFO

ifconfig enp0s20f6 up 169.254.9.102 netmask 255.255.255.0
./home/root/Servermodbus.py &
./home/root/main.py &
exit 0
```

Figura 11- Script de arranque.

En la figura 11 se observa el script que se ejecuta en el arranque y tiene como función fijar la IP estática del Intel Galileo e iniciar los scripts de servidor Modbus y principal. Al igual que en los script anteriores la primera línea le indica al sistema con cual programa debe interpretar el script, en este caso es Shell.

ANEXO III
INSTRUCTIVO DE CÓDIGO FUENTE LABVIEW

Este instructivo tiene como objetivo exponer las principales funciones utilizadas en el desarrollo del programa diseñado en el software LabVIEW para la ejecución del control supervisorio, el monitoreo de las variables en tiempo real, el registro de datos, el sistema de alarmas y seguridad de acceso.

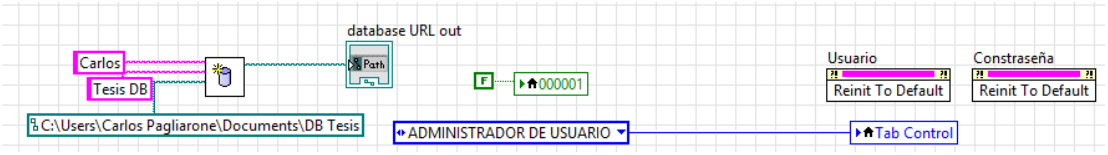

Figura 1. Inicialización del programa.

En la figura 1 se muestra la inicialización de variables donde se conecta el programa a la base de datos y se reestablece los valores de usuario y contraseña al valor por defecto, también se fija la pestaña "ADMINISTRADOR DE USUARIO" para que aparezca al inicio y se coloca un valor de falso a la bandera que se encarga de apagar el sistema embebido.

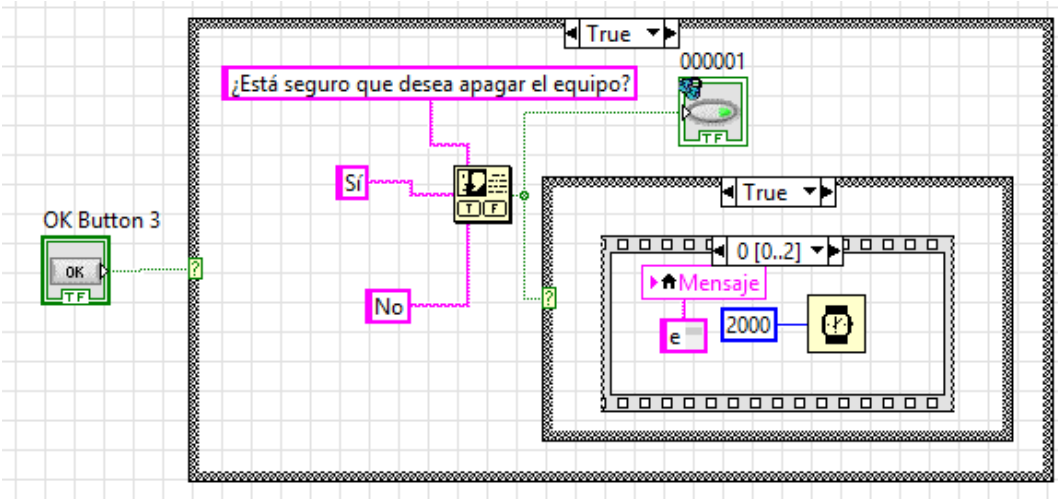

Figura 2. Proceso para apagar el sistema embebido.

En la figura 2 se muestra el procesa mediante el cual se apaga el sistema embebido. Al apretar el botón de “APAGAR INTEL GALILEO” se ejecuta el proceso dentro de la estrucutra case, en la que aparece un mensaje emergente para comprobar la selección, si el usuario efectivamente desea apagar el sistema, se activa la bobina 000001 del protocolo Modbus que se utiliza como bandera, se muestra un mensaje y finalmente se cierra la aplicación.

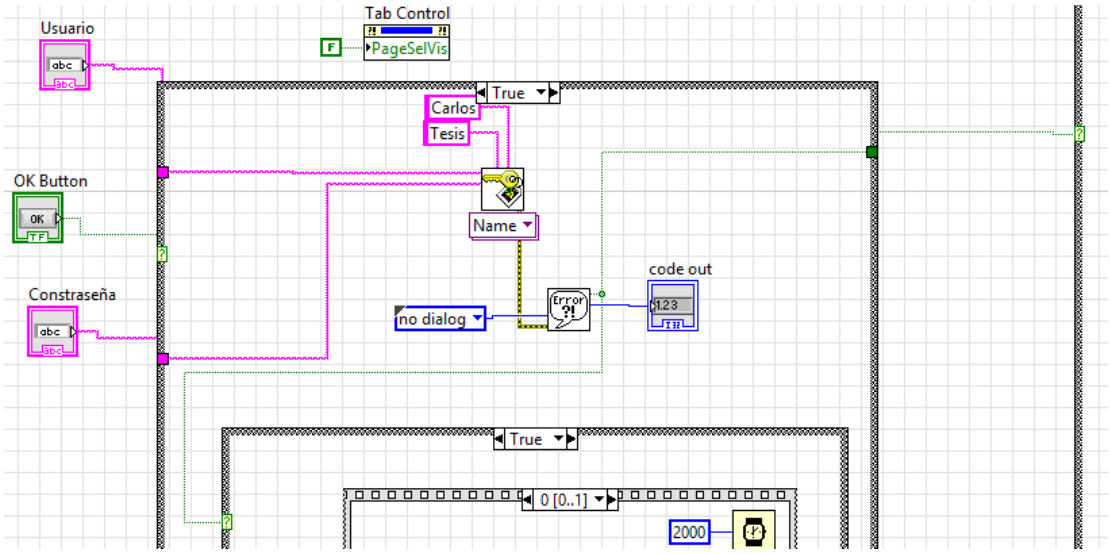

Figura 3. Proceso para iniciar session.

En la figura 3 se muestra el proceso para verificar el inicio de sesión. Primero se observa que el elemento “Tab Control” tiene la propiedad de visibilidad de las páginas desactivada para que el usuario no tenga acceso a las demás partes del sistema, a menos que inicie sesión con un usuario válido. Al apretar el botón de “INICIAR SESIÓN” el programa compara el usuario y la clave ingresados con la información que tiene en el sistema, si se produce un error muestra un mensaje indicando en cual campo fue, en caso de un inicio de sesión exitoso se muestra un mensaje que indica el usuario que acaba de iniciar sesión, se realiza la conexión con los “Trace” utilizados para el registro de datos y se inicia el ciclo de escaneo. En la figura 4 se muestra el programa que muestra el mensaje de inicio de sesión y la conexión con los “Trace”.

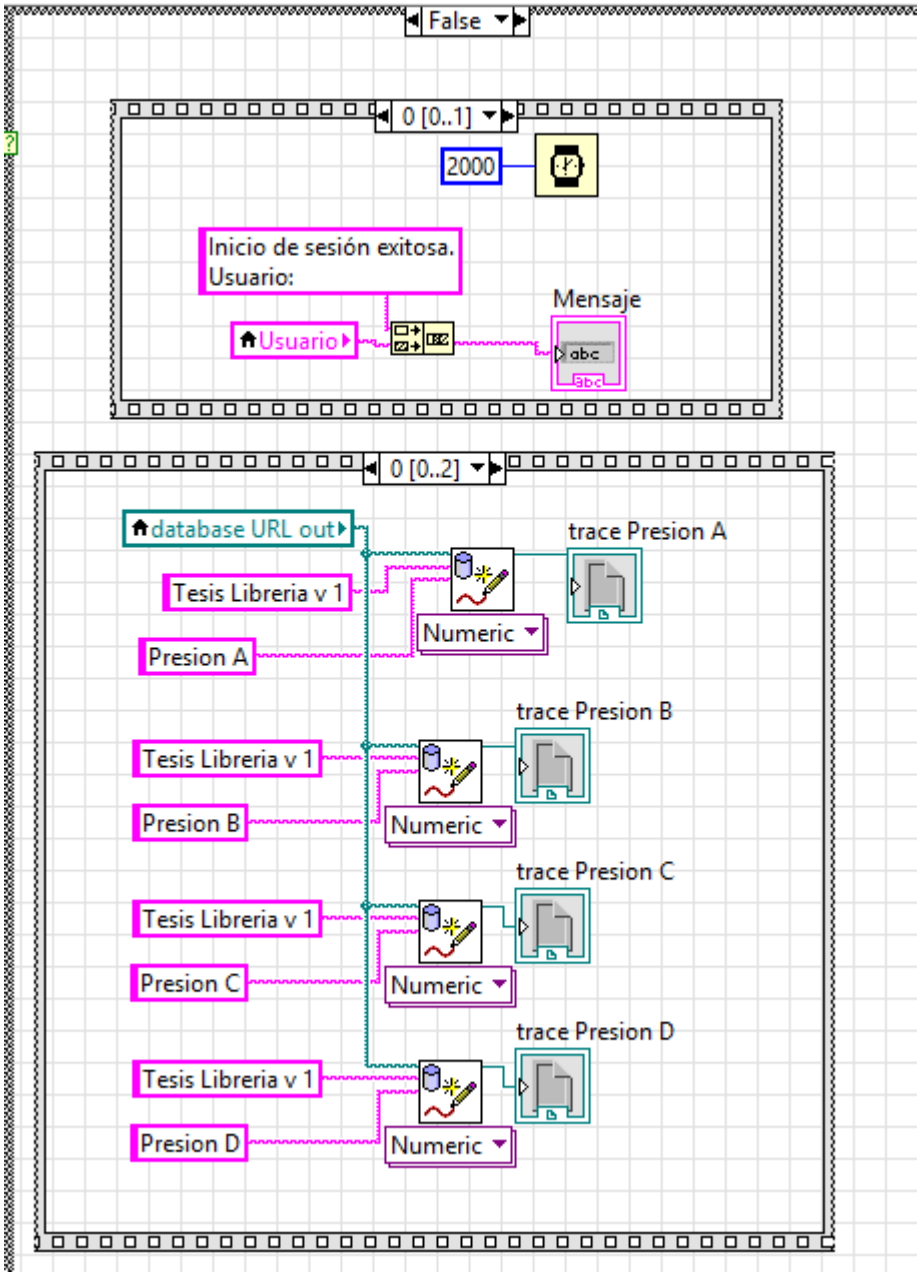

Figura 4. Programa que muestra el mensa de inicio de sesión y se conecta con los trace.

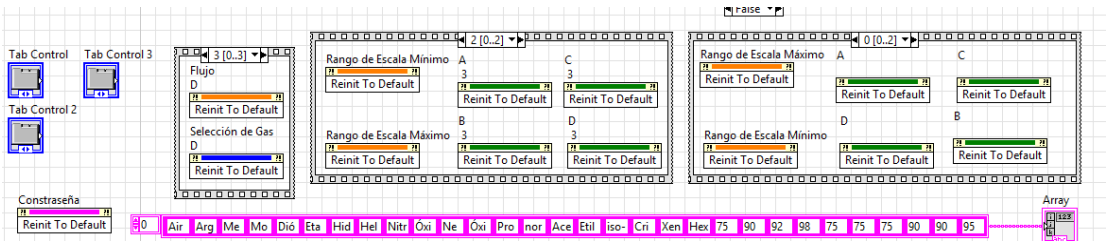

Figura 5. Inicialización de elementos de control.

En la figura 5 se muestra la inicialización a los valores por defecto de los elementos de control que fijan el punto de consigna, el gas a monitorear, el estados de los botones de visibilidad y controles de escala para las gráficas en tiempo real y registro de datos, además se observa un vector que contiene los gases que muestra el indicador de texto.

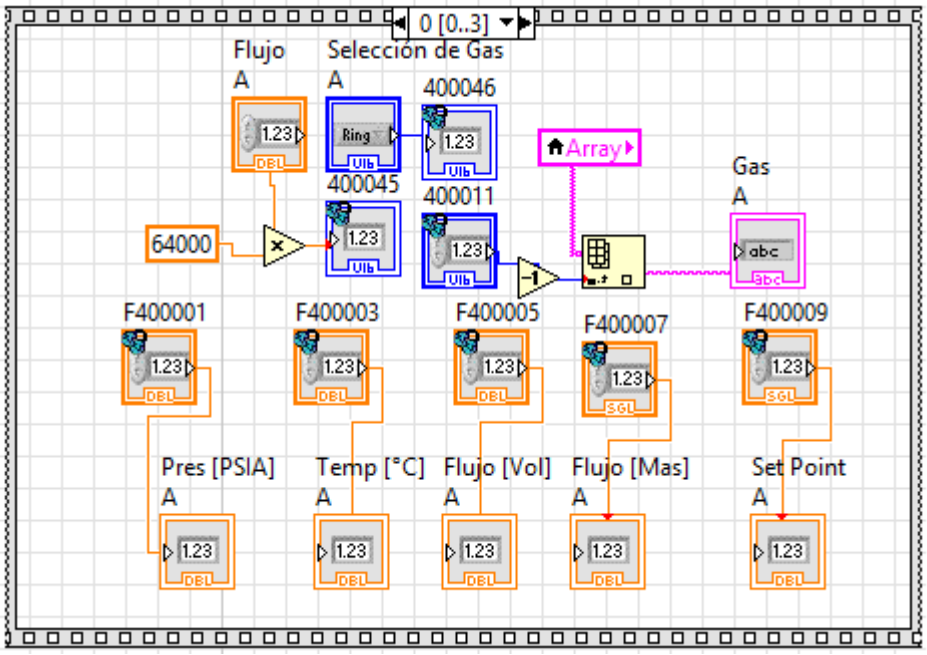

Figura 6. Proceso para escribir y leer del servidor Modbus.

En la figura 6 se observa el proceso mediante el cual el programa escribe y lee del servidor Modbus. Para realizar esto se utilizó la característica de I/O Server de LabVIEW creando la conexión a un servidor Modbus con el IP y el puerto

especificado en el Intel Galileo. A partir de dicho servidor se crearon para cada controlador de flujo 5 elementos doble Word (en la imagen F400001…F400009) a partir de los cuales se adquieren los datos del controlador y 1 registro Word (en la imagen 400011) para representar el gas que se está monitorizando, así mismo se utilizan 2 registros Words (en la imagen 400045 y 400046) para fijar el punto de consiga y seleccionar el gas.

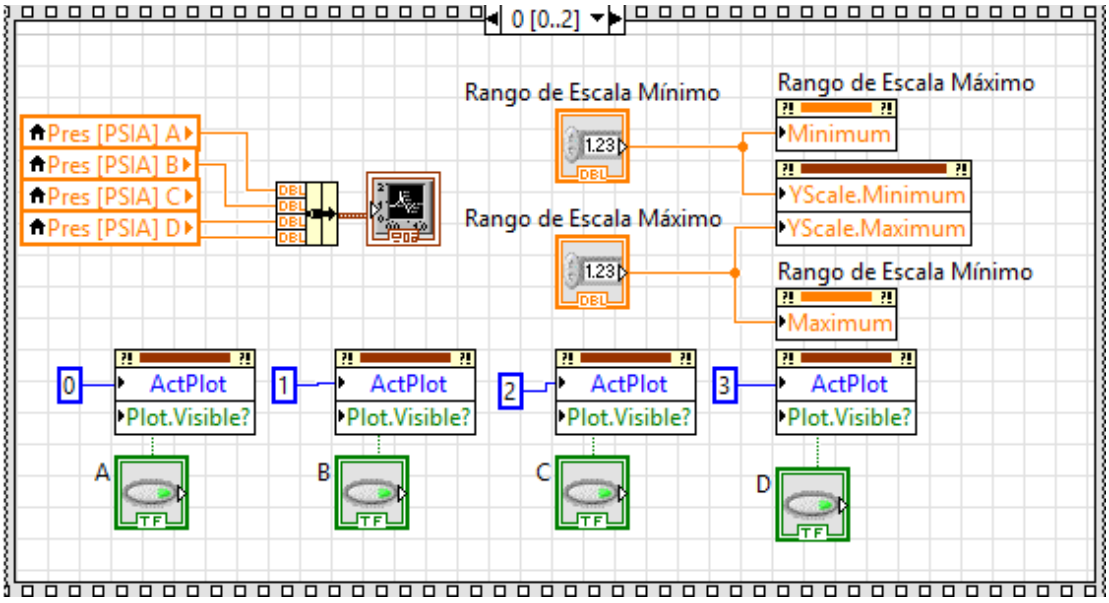

Figura 7. Programa para graficar el estado de las variables en tiempo real.

En la figura 7 se observa el programa que tiene como función leer los valores de los parámetros y graficarlos en tiempo real. En el programa se observa cómo se utiliza el bloque "Bundle" para agrupar las señales de los cuatro controladores y el resultado se ingresa al bloque de la gráfica. Además se observan varias propiedades del bloque de gráfica que se utilizan para controlar la visibilidad de las señales y la escala del gráfico, a través de los controles booleanos y numéricos respectivamente.

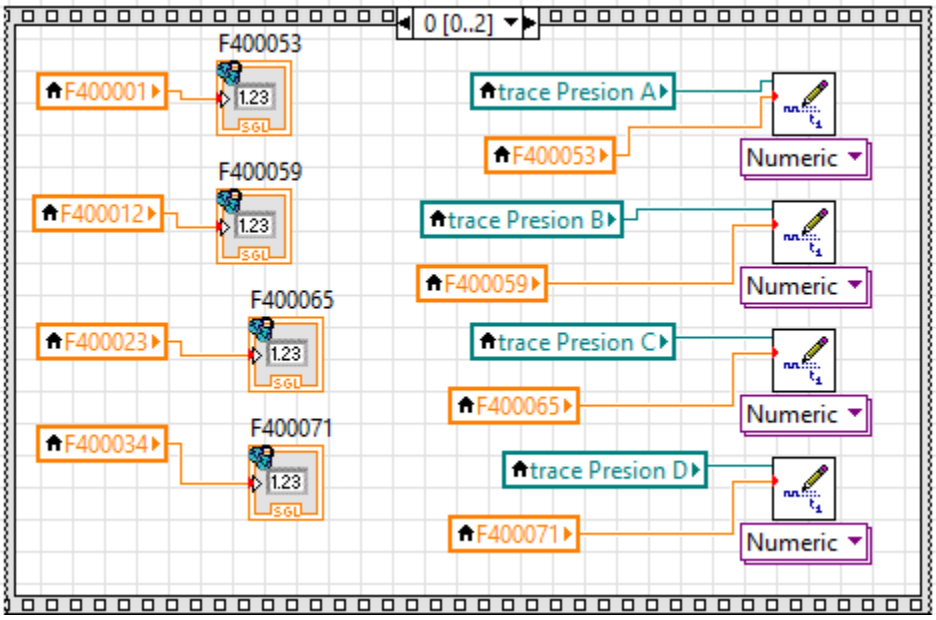

Figura 8. Programa para registrar los parámetros en la base de datos.

En la figura 8 se observa el programa en donde se crean copias (F400053, F400059, F400065 y F400071) de los elementos que muestran los parámetros de los controladores y se utilizan las copias para escribir en los "Trace", a través de los que se registran los valores en el histórico. Es necesario crear registros copias debido a que los elementos creados son locales con el motivo de mostrar su comportamiento en tiempo real, pero para almacenar la información en la base de datos a través de los "Trace" las variables tienen que ser públicas.

En la figura 9 se observa el bloque de histórico del módulo DSC a través del cual se seleccionan los "Trace" que se desean mostrar en la gráfica y, al igual que en la gráfica en tiempo real, se observan las propiedades de la gráfica XY que permiten cambiar la visibilidad de las curvas y modificar la escala.

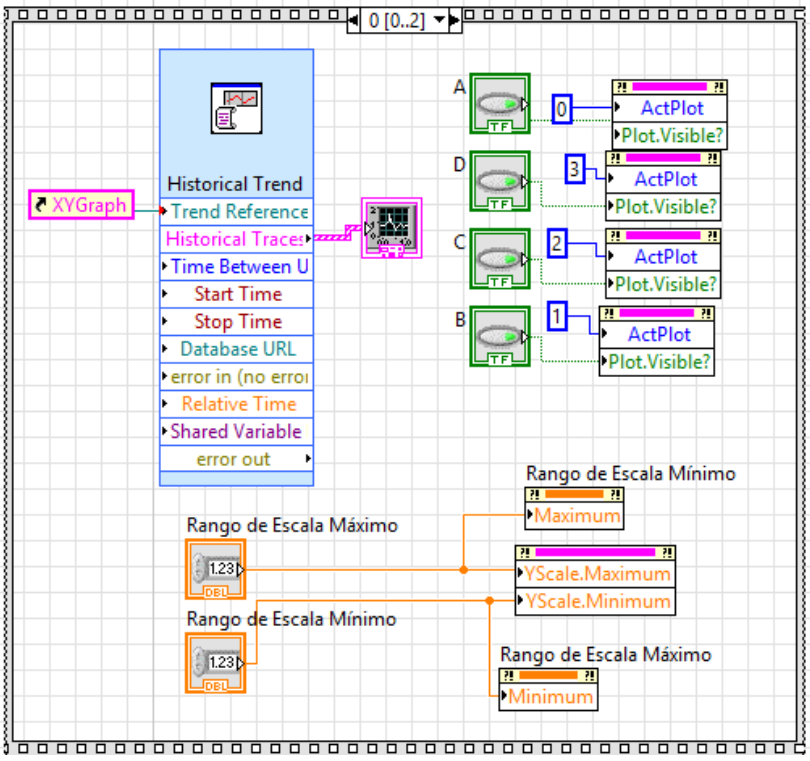

Figura 9. Programa para la gráfica de registro de datos.

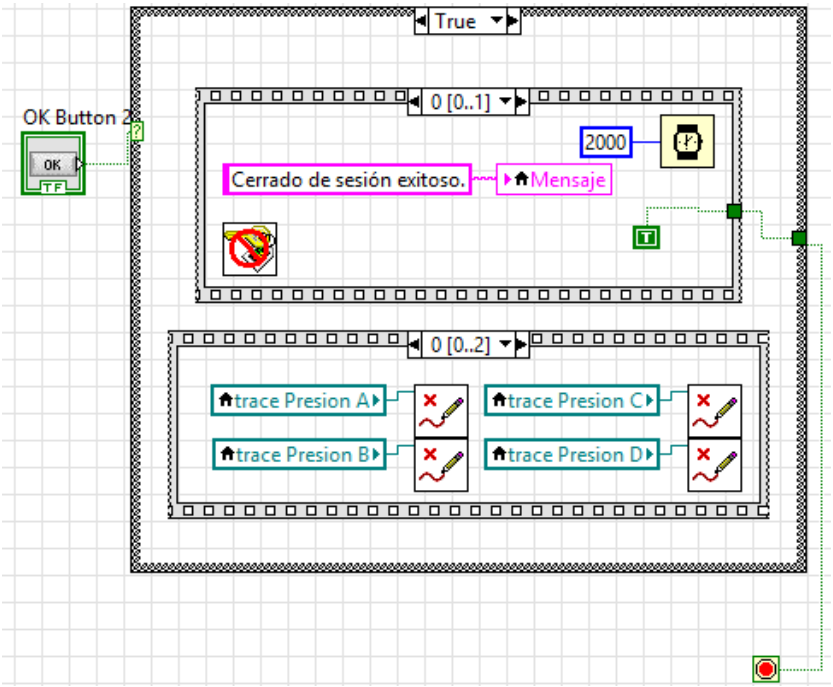

Figura 10. Programa para cerrar sesión.

En la figura 10 se observa el programa para cerrar sesión. Cuando se presiona el botón “CERRAR SESIÓN” se ejecutan las instrucciones dentro del “case”, las cuales muestran un mensaje, cierran las conexiones con los “Trace” y terminan el ciclo de escaneo, desactivando nuevamente la visibilidad de las páginas del “Tab Control”.

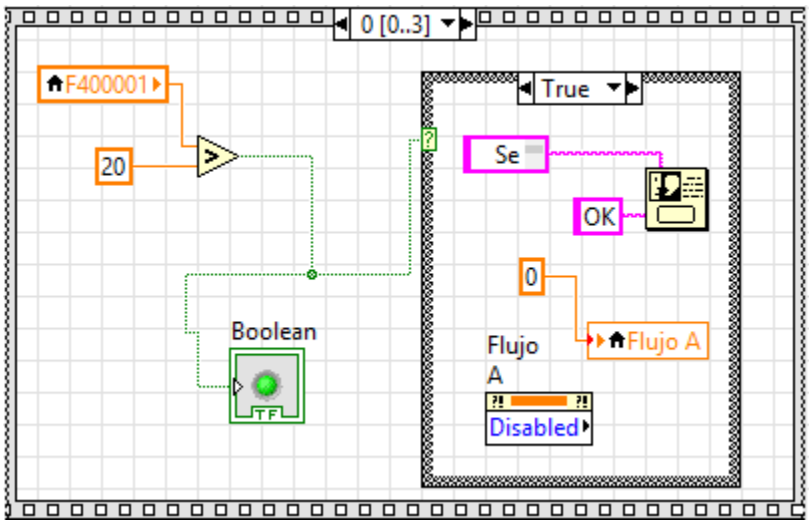

Figura 11. Proceso para activar la alarma a la salida del controlador.

En la figura 11 se muestra la activación de la alarma de seguridad que depende de la presión del controlador. Si la presión aumenta más de 20 PSIA se activa el LED, aparece un mensaje emergente, que no cesará hasta que la presión baje del valor establecido, fija un punto de consigna cero y deshabilita el elemento de control.

En la figura 12 se observa la activación de la alarma de seguridad que depende de flujo. En cada ciclo de escaneo se compara el punto de consigna con el flujo real medido, si durante un tiempo establecido el flujo real no alcanza el punto de consigna asignado se activa el LED, aparece un mensaje emergente, el cual no desaparecerá hasta s que se solucione la falla, y se coloca un punto de consiga cero para evitar sobrecalentamiento del equipo.

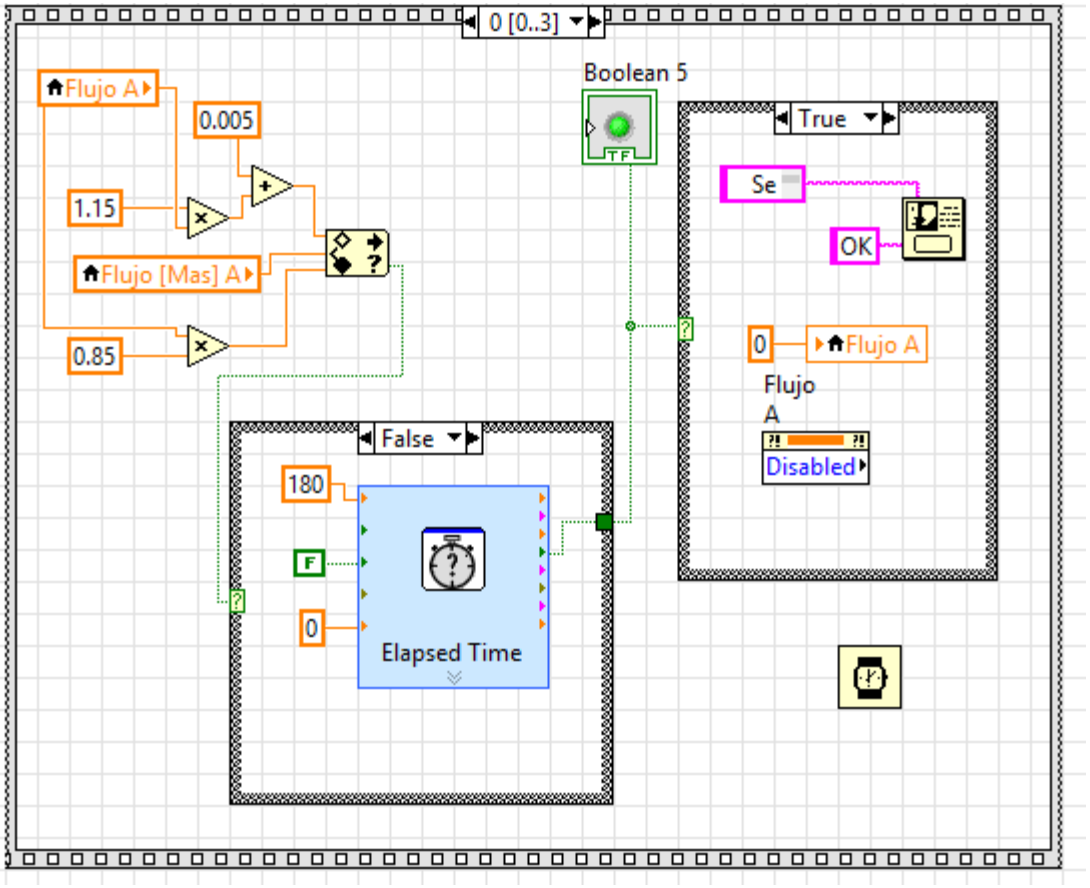

Figura 12. Programa para activar la alarma a la entrada del controlador.